CSE- 1006 LAB Assignment - 6

Academic year: 2020-2021 **Semester :** WIN

Faculty Name: Dr. Arun kumar Gopu **Date:** 16/3/2022

Student name: M.Taran **Reg. no.:** 19BCE7346

Data Manipulation with dplyr

The dplyr package is one of the most powerful and popular packages in R.

This package was written by the most popular R programmer Hadley Wickham who has written many useful R packages such as ggplot2, tidyr etc. This post includes several examples and tips of how to use the dplyr package for cleaning and transforming data. It's a complete tutorial on data manipulation and data wrangling with R.

What is dplyr?

The dplyr is a powerful R-package to manipulate, clean and summarize unstructured data. In short, it makes data exploration and data manipulation easy and fast in R.

What's special about dplyr?

The package "dplyr" comprises many functions that perform mostly used data manipulation operations such as applying filters, selecting specific columns, sorting data, adding or deleting columns and aggregating data.

Another most important advantage of this package is that it's very easy to learn and use dplyr functions. Also easy to recall these functions. For example, **filter()** is used to filter rows.

dplyr vs. Base R Functions

dplyr functions process faster than base R functions. It is because dplyr functions were written in a computationally efficient manner. They are also more stable in the syntax and better supports data frames than vectors.

SQL Queries vs. dplyr

People have been utilizing SQL for analyzing data for decades. Every modern data analysis software such as Python, R, SAS etc supports SQL commands. But SQL was never designed to perform data analysis. It was rather designed for querying and managing data. There are many data analysis operations where SQL fails or makes simple things difficult. For example, calculating median for multiple variables, converting wide format data to long format etc. Whereas, dplyr package was designed to do data

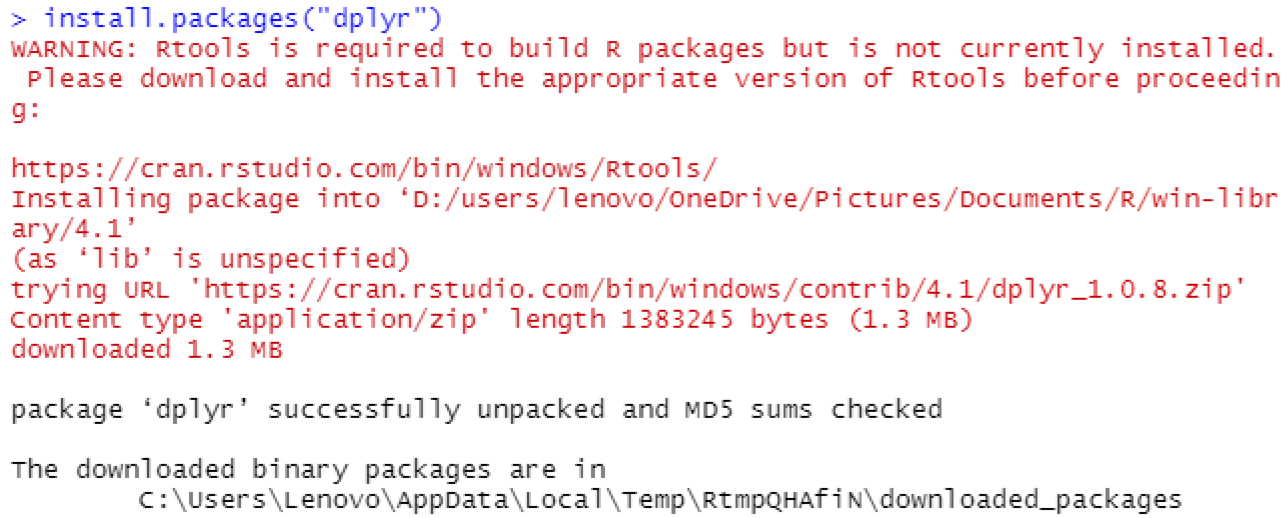
analysis.

The names of dplyr functions are similar to SQL commands such as **select()** for selecting variables, **group_by()** - group data by grouping variable, join() - joining two data sets. Also includes **inner_join()** and **left_join()**. It also supports subqueries for which SQL was popular for.

How to install and load dplyr package

To install the dplyr package, type the following command.

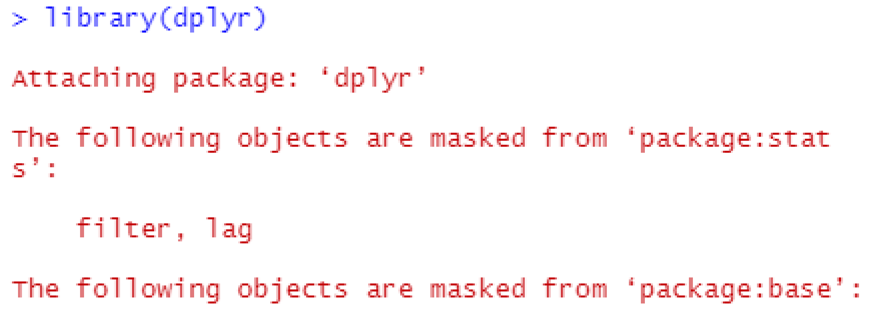
install.packages("dplyr")



```
> install.packages("dplyr")
WARNING: Rtools is required to build R packages but is not currently installed.
 Please download and install the appropriate version of Rtools before proceedin
g:

https://cran.rstudio.com/bin/windows/Rtools/
Installing package into ‘D:/users/lenovo/OneDrive/Pictures/Documents/R/win-libr
ary/4.1’
(as ‘lib’ is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.1/dplyr_1.0.8.zip'
Content type 'application/zip' length 1383245 bytes (1.3 MB)
downloaded 1.3 MB

package ‘dplyr’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\Lenovo\AppData\Local\Temp\RtmpQHAfiN\downloaded_packages
```

To load dplyr package, type the command below

library(dplyr)



```
> library(dplyr)

Attaching package: ‘dplyr’

The following objects are masked from ‘package:stat
s’:

    filter, lag

The following objects are masked from ‘package:base’:
```

Important dplyr Functions to remember

dplyr Function Description Equivalent SQL

select() Selecting columns (variables) SELECT

filter() Filter (subset) rows. WHERE

group_by() Group the data GROUP BY

summarise() Summarise (or aggregate) data

arrange() Sort the data ORDER BY

join() Joining data frames (tables) JOIN

mutate() Creating New Variables COLUMN

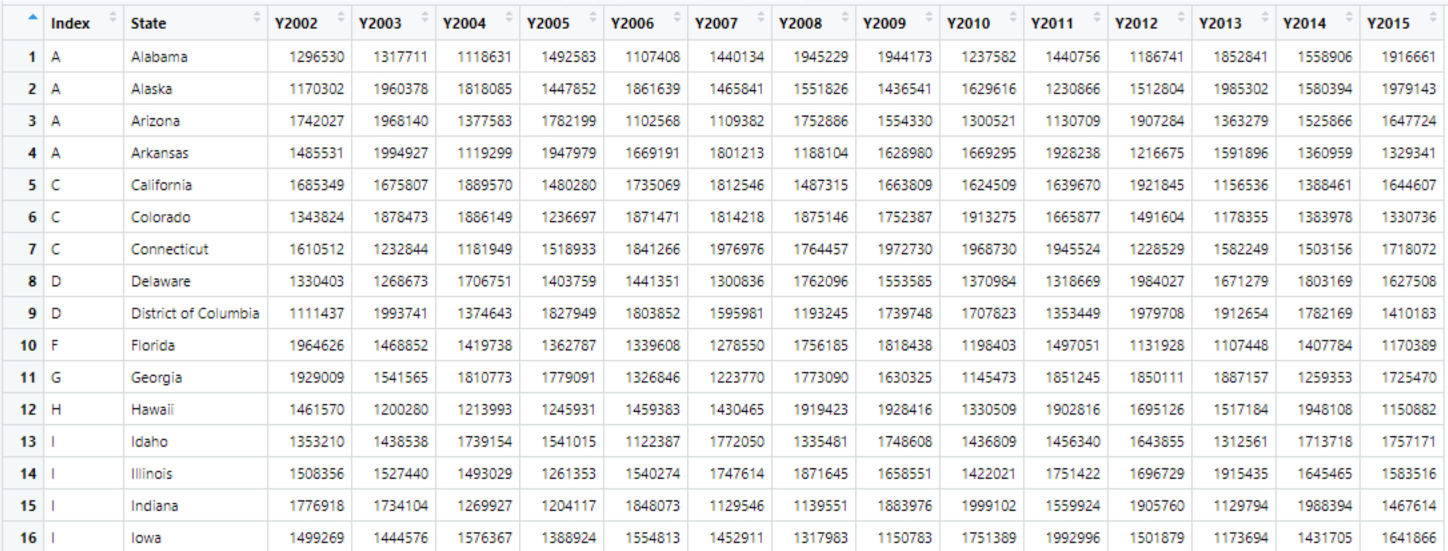
ALIAS

Data : Income Data by States

In this tutorial, we are using the following data which contains income generated by states from 2002 to 2015. **Note :** This data do not contain actual income figures of the states.

This dataset contains 51 observations (rows) and 16 variables (columns).

The snapshot of the first 6 rows of the dataset is shown below.



	Index	State	Y2002	Y2003	Y2004	Y2005	Y2006	Y2007	Y2008	Y2009	Y2010	Y2011	Y2012	Y2013	Y2014	Y2015
1	A	Alabama	1296530	1317711	1118631	1492583	1107408	1440134	1945229	1944173	1237582	1440756	1186741	1852841	1558906	1916661
2	A	Alaska	1170302	1960378	1818085	1447852	1861639	1465841	1551826	1436541	1629616	1230866	1512804	1985302	1580394	1979143
3	A	Arizona	1742027	1968140	1377583	1782199	1102568	1109382	1752886	1554330	1300521	1130709	1907284	1363279	1525866	1647724
4	A	Arkansas	1485531	1994927	1119299	1947979	1669191	1801213	1188104	1628980	1669295	1928238	1216675	1591896	1360959	1329341
5	C	California	1685349	1675807	1889570	1480280	1735069	1812546	1487315	1663809	1624509	1639670	1921845	1156536	1388461	1644607
6	C	Colorado	1343824	1878473	1886149	1236697	1871471	1814218	1875146	1752387	1913275	1665877	1491604	1178355	1383978	1330736
7	C	Connecticut	1610512	1232844	1181949	1518933	1841266	1976976	1764457	1972730	1968730	1945524	1228529	1582249	1503156	1718072
8	D	Delaware	1330403	1268673	1706751	1403759	1441351	1300836	1762096	1553585	1370984	1318669	1984027	1671279	1803169	1627508
9	D	District of Columbia	1111437	1993741	1374643	1827949	1803852	1595981	1193245	1739748	1707823	1353449	1979708	1912654	1782169	1410183
10	F	Florida	1964626	1468852	1419738	1362787	1339608	1278550	1756185	1818438	1198403	1497051	1131928	1107448	1407784	1170389
11	G	Georgia	1929009	1541565	1810773	1779091	1326846	1223770	1773090	1630325	1145473	1851245	1850111	1887157	1259353	1725470
12	H	Hawaii	1461570	1200280	1213993	1245931	1459383	1430465	1919423	1928416	1330509	1902816	1695126	1517184	1948108	1150882
13	I	Idaho	1353210	1438538	1739154	1541015	1122387	1772050	1335481	1748608	1436809	1456340	1643855	1312561	1713718	1757171
14	I	Illinois	1508356	1527440	1493029	1261353	1540274	1747614	1871645	1658551	1422021	1751422	1696729	1915435	1645465	1583516
15	I	Indiana	1776918	1734104	1269927	1204117	1848073	1129546	1139551	1883976	1999102	1559924	1905760	1129794	1988394	1467614
16	I	Iowa	1499269	1444576	1576367	1388924	1554813	1452911	1317983	1150783	1751389	1992996	1501879	1173694	1431705	1641866

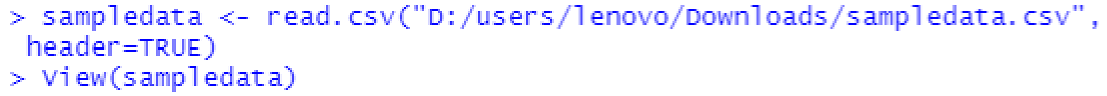
Download the Dataset

How to load Data

Submit the following code. ***Change the file path in the code below.***

```
sampledata <-
read.csv("D:/users/lenovo/Downloads/sampledata.csv",
```

header=TRUE)

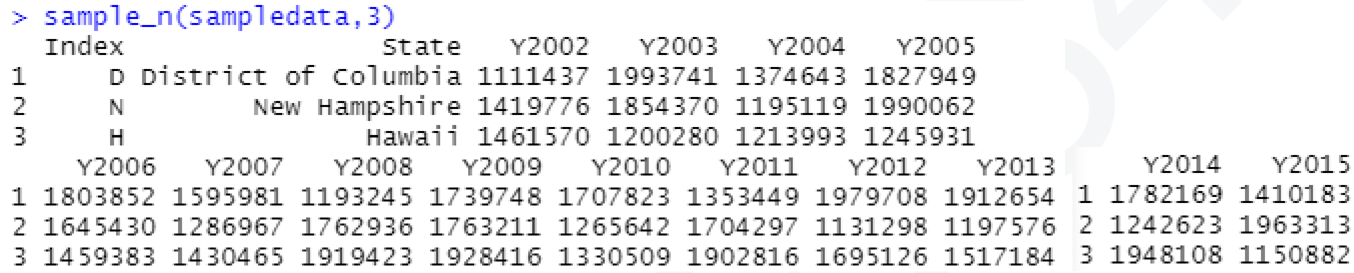


```
> sampledata <- read.csv("D:/users/lenovo/Downloads/sampledata.csv",
 header=TRUE)
> View(sampledata)
```

Example 1 : Selecting Random N Rows

The **sample_n** function selects random rows from a data frame (or table). The second parameter of the function tells R the number of rows to select.

sample_n(sampledata,3)

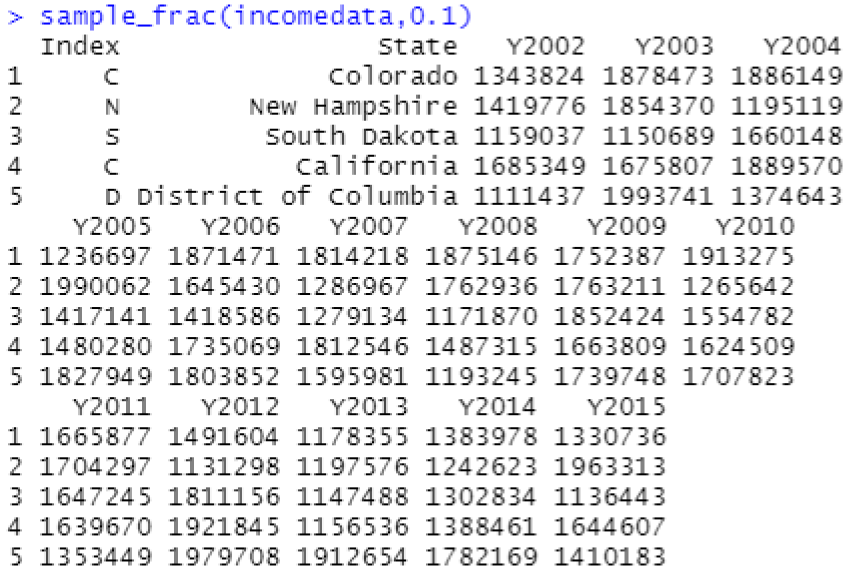


```
> sample_n(sampledata,3)
  Index                State   Y2002   Y2003   Y2004   Y2005
1     D District of Columbia 1111437 1993741 1374643 1827949
2     N        New Hampshire 1419776 1854370 1195119 1990062
3     H               Hawaii 1461570 1200280 1213993 1245931
    Y2006   Y2007   Y2008   Y2009   Y2010   Y2011   Y2012   Y2013     Y2014   Y2015
1 1803852 1595981 1193245 1739748 1707823 1353449 1979708 1912654 1 1782169 1410183
2 1645430 1286967 1762936 1763211 1265642 1704297 1131298 1197576 2 1242623 1963313
3 1459383 1430465 1919423 1928416 1330509 1902816 1695126 1517184 3 1948108 1150882
```

Example 2 : Selecting Random Fraction of Rows

The **sample_frac** function returns randomly N% of rows. In the example below, it returns randomly 10% of rows.

sample_frac(incomedata,0.1)



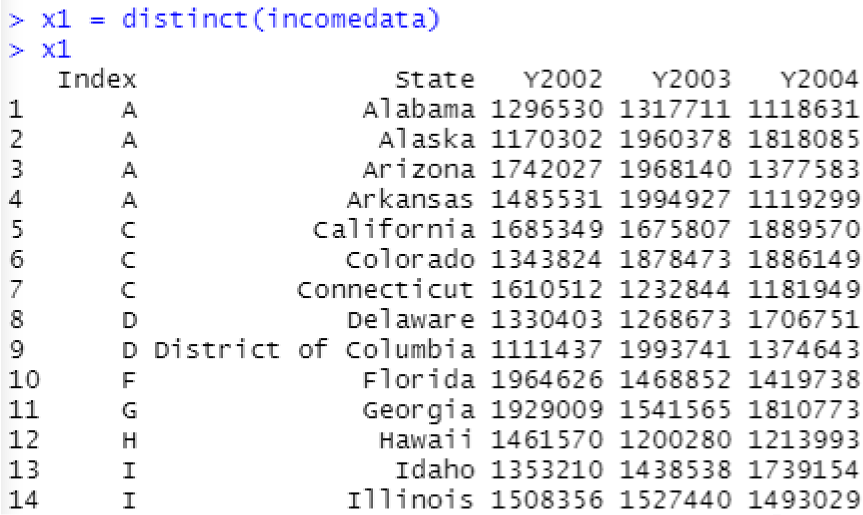
```
> sample_frac(incomedata,0.1)
  Index                State   Y2002   Y2003   Y2004
1     C             Colorado 1343824 1878473 1886149
2     N        New Hampshire 1419776 1854370 1195119
3     S         South Dakota 1159037 1150689 1660148
4     C           California 1685349 1675807 1889570
5     D District of Columbia 1111437 1993741 1374643
    Y2005   Y2006   Y2007   Y2008   Y2009   Y2010
1 1236697 1871471 1814218 1875146 1752387 1913275
2 1990062 1645430 1286967 1762936 1763211 1265642
3 1417141 1418586 1279134 1171870 1852424 1554782
4 1480280 1735069 1812546 1487315 1663809 1624509
5 1827949 1803852 1595981 1193245 1739748 1707823
    Y2011   Y2012   Y2013   Y2014   Y2015
1 1665877 1491604 1178355 1383978 1330736
2 1704297 1131298 1197576 1242623 1963313
3 1647245 1811156 1147488 1302834 1136443
4 1639670 1921845 1156536 1388461 1644607
5 1353449 1979708 1912654 1782169 1410183
```

Example 3 : Remove Duplicate Rows based on all the variables

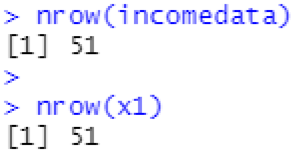
(Complete Row)

The **distinct function** is used to eliminate duplicates.

x1 = distinct(incomedata)



```
> x1 = distinct(incomedata)
> x1
   Index                State   Y2002   Y2003   Y2004
1      A              Alabama 1296530 1317711 1118631
2      A               Alaska 1170302 1960378 1818085
3      A              Arizona 1742027 1968140 1377583
4      A             Arkansas 1485531 1994927 1119299
5      C           California 1685349 1675807 1889570
6      C             Colorado 1343824 1878473 1886149
7      C          Connecticut 1610512 1232844 1181949
8      D             Delaware 1330403 1268673 1706751
9      D District of Columbia 1111437 1993741 1374643
10     F              Florida 1964626 1468852 1419738
11     G              Georgia 1929009 1541565 1810773
12     H               Hawaii 1461570 1200280 1213993
13     I                Idaho 1353210 1438538 1739154
14     I             Illinois 1508356 1527440 1493029
```

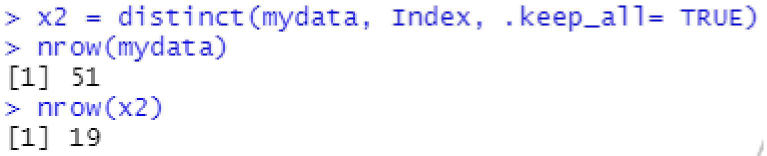


```
> nrow(incomedata)
[1] 51
>
> nrow(x1)
[1] 51
```

In this dataset, there is not a single duplicate row so it returned the same number of rows as in mydata.

Example 4 : Remove Duplicate Rows based on a variable The **.keep_all** function is used to retain all other variables in the output data frame.

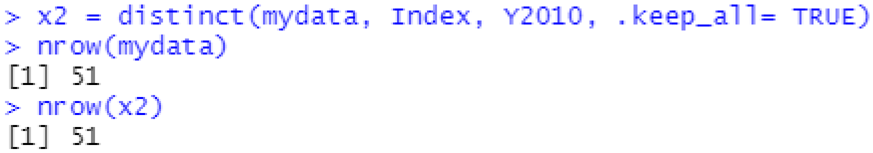
x2 = distinct(mydata, Index, **.keep_all= TRUE**)



```
> x2 = distinct(mydata, Index, .keep_all= TRUE)
> nrow(mydata)
[1] 51
> nrow(x2)
[1] 19
```

Example 5 : Remove Duplicates Rows based on multiple variables In the example below, we are using two variables - **Index, Y2010** to determine uniqueness.

x2 = distinct(mydata, **Index**, **Y2010**, .keep_all= TRUE)



```
> x2 = distinct(mydata, Index, Y2010, .keep_all= TRUE)
> nrow(mydata)
[1] 51
> nrow(x2)
[1] 51
```

select( ) Function

It is used to select only desired variables.

select() syntax : select(data , ...)

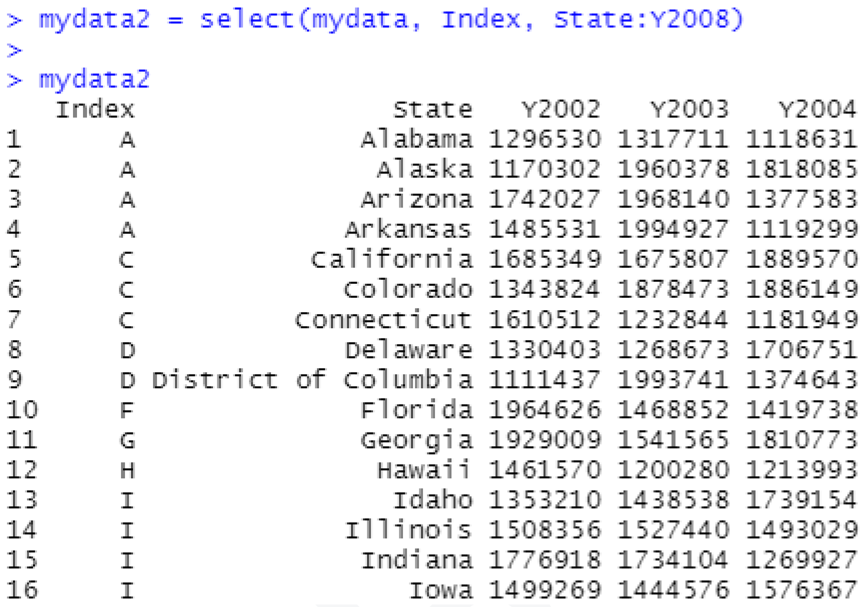
data : Data Frame

.... : Variables by name or by function

Example 6 : Selecting Variables (or Columns)

Suppose you are asked to select only a few variables. The code below selects variables "Index", columns from "State" to "Y2008".

mydata2 = select(mydata, Index, State:Y2008)

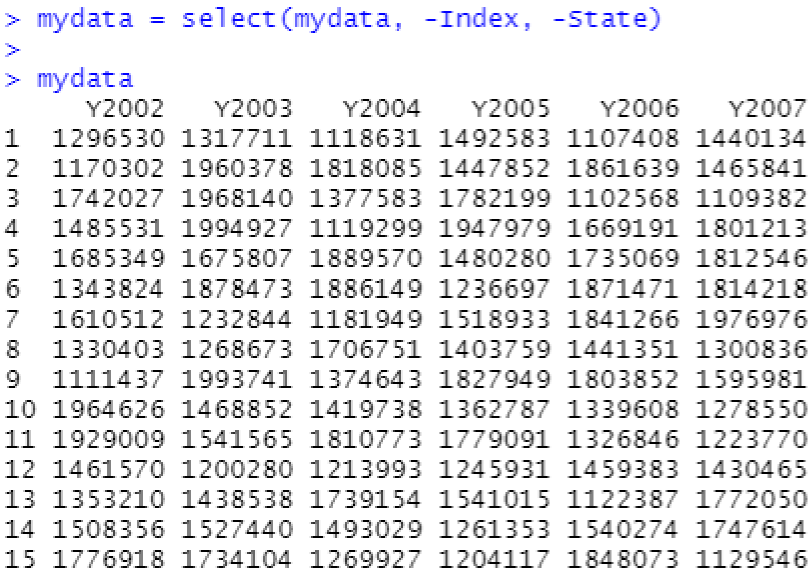


```
> mydata2 = select(mydata, Index, State:Y2008)
>
> mydata2
   Index                State   Y2002   Y2003   Y2004
1      A              Alabama 1296530 1317711 1118631
2      A               Alaska 1170302 1960378 1818085
3      A              Arizona 1742027 1968140 1377583
4      A             Arkansas 1485531 1994927 1119299
5      C           California 1685349 1675807 1889570
6      C             Colorado 1343824 1878473 1886149
7      C          Connecticut 1610512 1232844 1181949
8      D             Delaware 1330403 1268673 1706751
9      D District of Columbia 1111437 1993741 1374643
10     F              Florida 1964626 1468852 1419738
11     G              Georgia 1929009 1541565 1810773
12     H               Hawaii 1461570 1200280 1213993
13     I                Idaho 1353210 1438538 1739154
14     I             Illinois 1508356 1527440 1493029
15     I              Indiana 1776918 1734104 1269927
16     I                 Iowa 1499269 1444576 1576367
```

Example 7 : Dropping Variables

The **minus sign** before a variable tells R to drop the variable.

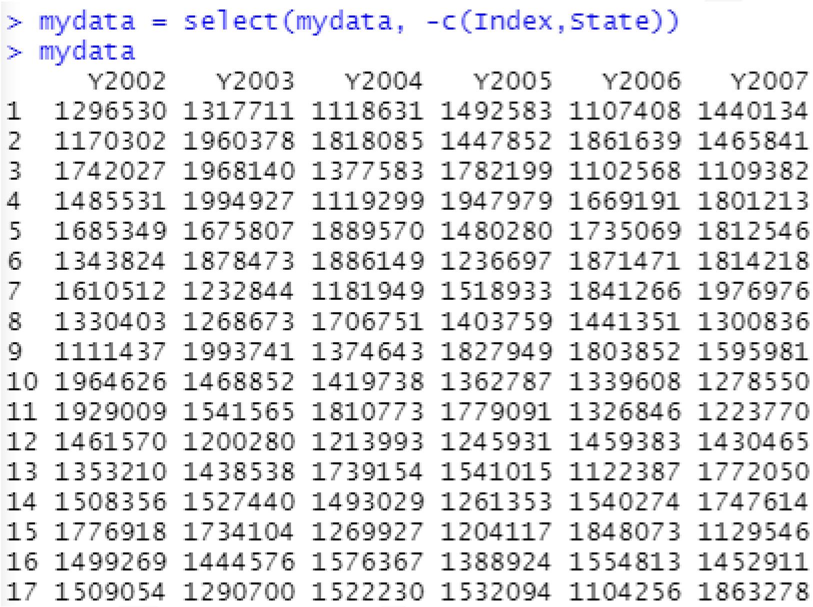
mydata = select(mydata, -Index, -State)



```
> mydata = select(mydata, -Index, -State)
>
> mydata
     Y2002   Y2003   Y2004   Y2005   Y2006   Y2007
1  1296530 1317711 1118631 1492583 1107408 1440134
2  1170302 1960378 1818085 1447852 1861639 1465841
3  1742027 1968140 1377583 1782199 1102568 1109382
4  1485531 1994927 1119299 1947979 1669191 1801213
5  1685349 1675807 1889570 1480280 1735069 1812546
6  1343824 1878473 1886149 1236697 1871471 1814218
7  1610512 1232844 1181949 1518933 1841266 1976976
8  1330403 1268673 1706751 1403759 1441351 1300836
9  1111437 1993741 1374643 1827949 1803852 1595981
10 1964626 1468852 1419738 1362787 1339608 1278550
11 1929009 1541565 1810773 1779091 1326846 1223770
12 1461570 1200280 1213993 1245931 1459383 1430465
13 1353210 1438538 1739154 1541015 1122387 1772050
14 1508356 1527440 1493029 1261353 1540274 1747614
15 1776918 1734104 1269927 1204117 1848073 1129546
```

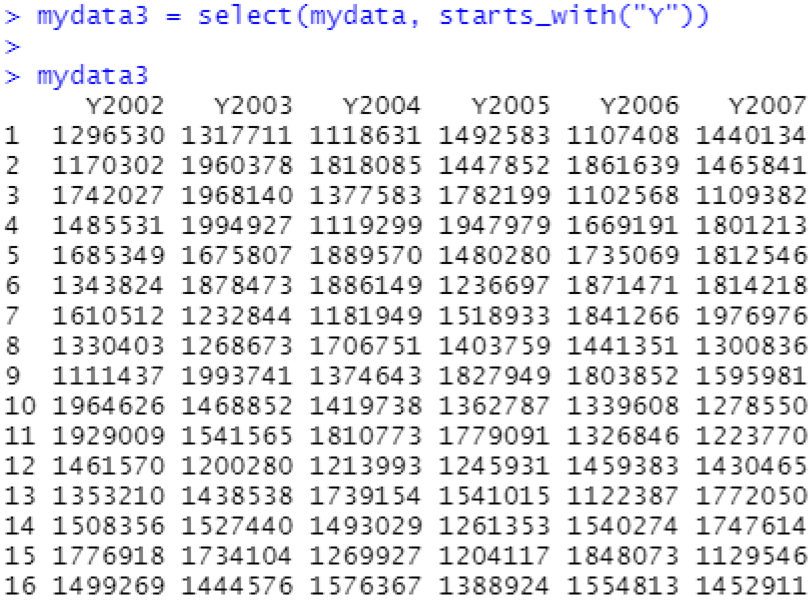
The above code can also be written like :

mydata = select(mydata, -c(Index,State))



```
> mydata = select(mydata, -c(Index,State))
> mydata
     Y2002   Y2003   Y2004   Y2005   Y2006   Y2007
1  1296530 1317711 1118631 1492583 1107408 1440134
2  1170302 1960378 1818085 1447852 1861639 1465841
3  1742027 1968140 1377583 1782199 1102568 1109382
4  1485531 1994927 1119299 1947979 1669191 1801213
5  1685349 1675807 1889570 1480280 1735069 1812546
6  1343824 1878473 1886149 1236697 1871471 1814218
7  1610512 1232844 1181949 1518933 1841266 1976976
8  1330403 1268673 1706751 1403759 1441351 1300836
9  1111437 1993741 1374643 1827949 1803852 1595981
10 1964626 1468852 1419738 1362787 1339608 1278550
11 1929009 1541565 1810773 1779091 1326846 1223770
12 1461570 1200280 1213993 1245931 1459383 1430465
13 1353210 1438538 1739154 1541015 1122387 1772050
14 1508356 1527440 1493029 1261353 1540274 1747614
15 1776918 1734104 1269927 1204117 1848073 1129546
16 1499269 1444576 1576367 1388924 1554813 1452911
17 1509054 1290700 1522230 1532094 1104256 1863278
```

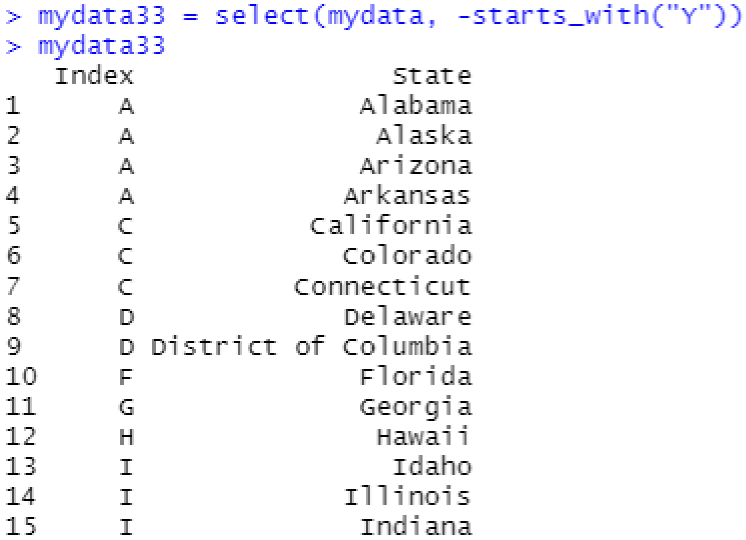
Example 8 : Selecting or Dropping Variables starts with 'Y' The **starts_with()** function is used to select variables that start with an alphabet. mydata3 = select(mydata, starts_with("Y"))



```
> mydata3 = select(mydata, starts_with("Y"))
>
> mydata3
     Y2002   Y2003   Y2004   Y2005   Y2006   Y2007
1  1296530 1317711 1118631 1492583 1107408 1440134
2  1170302 1960378 1818085 1447852 1861639 1465841
3  1742027 1968140 1377583 1782199 1102568 1109382
4  1485531 1994927 1119299 1947979 1669191 1801213
5  1685349 1675807 1889570 1480280 1735069 1812546
6  1343824 1878473 1886149 1236697 1871471 1814218
7  1610512 1232844 1181949 1518933 1841266 1976976
8  1330403 1268673 1706751 1403759 1441351 1300836
9  1111437 1993741 1374643 1827949 1803852 1595981
10 1964626 1468852 1419738 1362787 1339608 1278550
11 1929009 1541565 1810773 1779091 1326846 1223770
12 1461570 1200280 1213993 1245931 1459383 1430465
13 1353210 1438538 1739154 1541015 1122387 1772050
14 1508356 1527440 1493029 1261353 1540274 1747614
15 1776918 1734104 1269927 1204117 1848073 1129546
16 1499269 1444576 1576367 1388924 1554813 1452911
```

Adding a negative sign before starts_with() implies dropping the variables starts with 'Y'

mydata33 = select(mydata, -starts_with("Y"))



```
> mydata33 = select(mydata, -starts_with("Y"))
> mydata33
   Index                State
1      A              Alabama
2      A               Alaska
3      A              Arizona
4      A             Arkansas
5      C           California
6      C             Colorado
7      C          Connecticut
8      D             Delaware
9      D District of Columbia
10     F              Florida
11     G              Georgia
12     H               Hawaii
13     I                Idaho
14     I             Illinois
15     I              Indiana
```

The following functions help you to select variables based on their names.

Helpers Description

starts_with() Starts with a prefix

ends_with() Ends with a prefix

contains() Contains a literal string

matches() Matches a regular expression

Output

num_range() Numerical range like x01, x02,

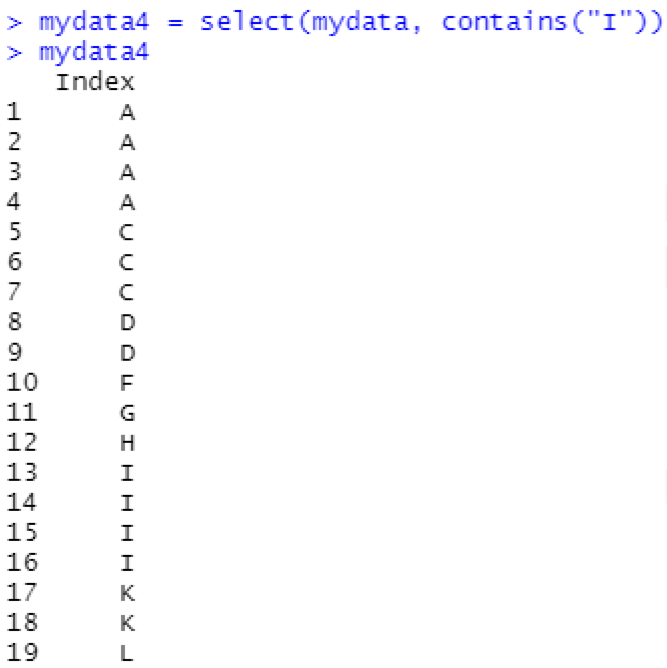
x03.

one_of() Variables in character vector.

everything() All variables.

Example 9 : Selecting Variables contain 'I' in their names

mydata4 = select(mydata, contains("I"))

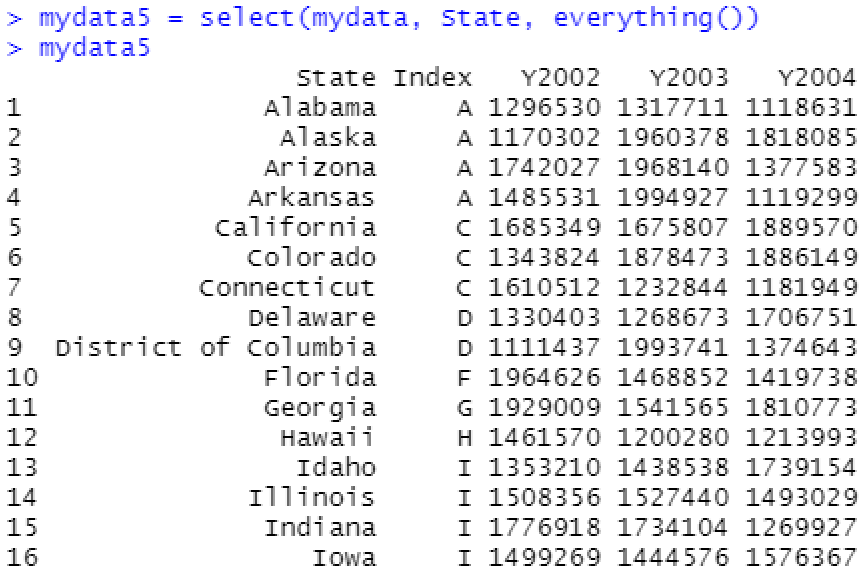


```
> mydata4 = select(mydata, contains("I"))
> mydata4
   Index
1      A
2      A
3      A
4      A
5      C
6      C
7      C
8      D
9      D
10     F
11     G
12     H
13     I
14     I
15     I
16     I
17     K
18     K
19     L
```

Example 10 : Reorder Variables

The code below keeps variable **'State'** in the front and the remaining variables follow that.

mydata5 = select(mydata, State, everything())



```
> mydata5 = select(mydata, State, everything())
> mydata5
                  State Index   Y2002   Y2003   Y2004
1               Alabama     A 1296530 1317711 1118631
2                Alaska     A 1170302 1960378 1818085
3               Arizona     A 1742027 1968140 1377583
4              Arkansas     A 1485531 1994927 1119299
5            California     C 1685349 1675807 1889570
6              Colorado     C 1343824 1878473 1886149
7           Connecticut     C 1610512 1232844 1181949
8              Delaware     D 1330403 1268673 1706751
9  District of Columbia     D 1111437 1993741 1374643
10              Florida     F 1964626 1468852 1419738
11              Georgia     G 1929009 1541565 1810773
12               Hawaii     H 1461570 1200280 1213993
13                Idaho     I 1353210 1438538 1739154
14             Illinois     I 1508356 1527440 1493029
15              Indiana     I 1776918 1734104 1269927
16                 Iowa     I 1499269 1444576 1576367
```

rename( ) Function

It is used to change variable names.

rename() syntax : rename(data , new_name = old_name)

data : Data Frame

new_name : New variable name you want to keep

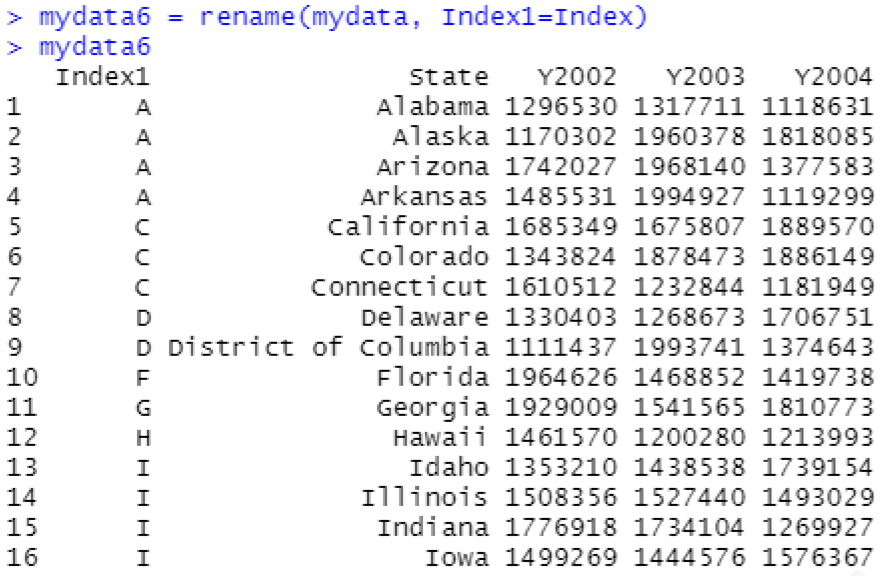
old_name : Existing Variable Name

Example 11 : Rename Variables

The rename function can be used to rename variables.

In the following code, we are renaming **'Index'** variable to **'Index1'**.

mydata6 = rename(mydata, Index1=Index)



```
> mydata6 = rename(mydata, Index1=Index)
> mydata6
   Index1                State   Y2002   Y2003   Y2004
1       A              Alabama 1296530 1317711 1118631
2       A               Alaska 1170302 1960378 1818085
3       A              Arizona 1742027 1968140 1377583
4       A             Arkansas 1485531 1994927 1119299
5       C           California 1685349 1675807 1889570
6       C             Colorado 1343824 1878473 1886149
7       C          Connecticut 1610512 1232844 1181949
8       D             Delaware 1330403 1268673 1706751
9       D District of Columbia 1111437 1993741 1374643
10      F              Florida 1964626 1468852 1419738
11      G              Georgia 1929009 1541565 1810773
12      H               Hawaii 1461570 1200280 1213993
13      I                Idaho 1353210 1438538 1739154
14      I             Illinois 1508356 1527440 1493029
15      I              Indiana 1776918 1734104 1269927
16      I                 Iowa 1499269 1444576 1576367
```

filter( ) Function

It is used to subset data with matching

logical conditions.

6/24

filter() syntax : filter(data , ...)

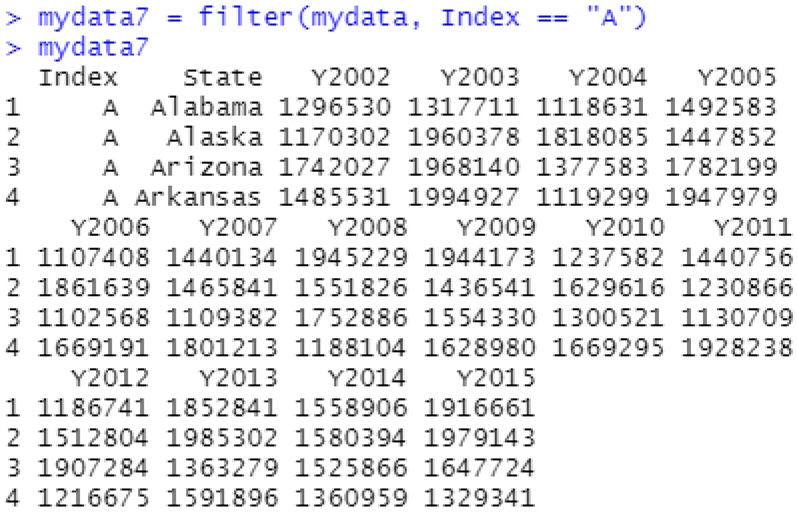
data : Data Frame

.... : Logical Condition

Example 12 : Filter Rows

Suppose you need to subset data. You want to filter rows and retain only those values in which Index is equal to A.

mydata7 = filter(mydata, Index == "A")



```
> mydata7 = filter(mydata, Index == "A")
> mydata7
  Index    State   Y2002   Y2003   Y2004   Y2005
1     A  Alabama 1296530 1317711 1118631 1492583
2     A   Alaska 1170302 1960378 1818085 1447852
3     A  Arizona 1742027 1968140 1377583 1782199
4     A Arkansas 1485531 1994927 1119299 1947979
    Y2006   Y2007   Y2008   Y2009   Y2010   Y2011
1 1107408 1440134 1945229 1944173 1237582 1440756
2 1861639 1465841 1551826 1436541 1629616 1230866
3 1102568 1109382 1752886 1554330 1300521 1130709
4 1669191 1801213 1188104 1628980 1669295 1928238
    Y2012   Y2013   Y2014   Y2015
1 1186741 1852841 1558906 1916661
2 1512804 1985302 1580394 1979143
3 1907284 1363279 1525866 1647724
4 1216675 1591896 1360959 1329341
```

Example 13 : Multiple Selection Criteria

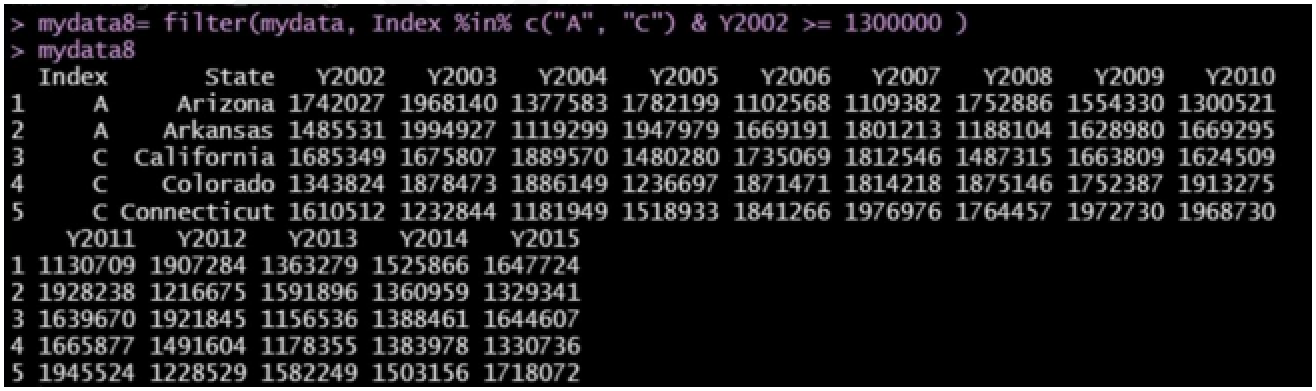
The **%in%** operator can be used to select multiple items. In the following program, we are telling R to select rows against 'A' and 'C' in column 'Index'.

mydata7 = filter(mydata6, Index %in% c("A", "C"))

Example 14 : 'AND' Condition in Selection Criteria

Suppose you need to apply 'AND' condition. In this case, we are picking data for 'A' and 'C' in the column 'Index' and income greater than 1.3 million in Year 2002.

mydata8 = filter(mydata6, Index %in% c("A", "C") & Y2002 >= 1300000)



```
> mydata8= filter(mydata, Index %in% c("A", "C") & Y2002 >= 1300000 )
> mydata8
  Index       State   Y2002   Y2003   Y2004   Y2005   Y2006   Y2007   Y2008   Y2009   Y2010
1     A     Arizona 1742027 1968140 1377583 1782199 1102568 1109382 1752886 1554330 1300521
2     A    Arkansas 1485531 1994927 1119299 1947979 1669191 1801213 1188104 1628980 1669295
3     C  California 1685349 1675807 1889570 1480280 1735069 1812546 1487315 1663809 1624509
4     C    Colorado 1343824 1878473 1886149 1236697 1871471 1814218 1875146 1752387 1913275
5     C Connecticut 1610512 1232844 1181949 1518933 1841266 1976976 1764457 1972730 1968730
    Y2011   Y2012   Y2013   Y2014   Y2015
1 1130709 1907284 1363279 1525866 1647724
2 1928238 1216675 1591896 1360959 1329341
3 1639670 1921845 1156536 1388461 1644607
4 1665877 1491604 1178355 1383978 1330736
5 1945524 1228529 1582249 1503156 1718072
```

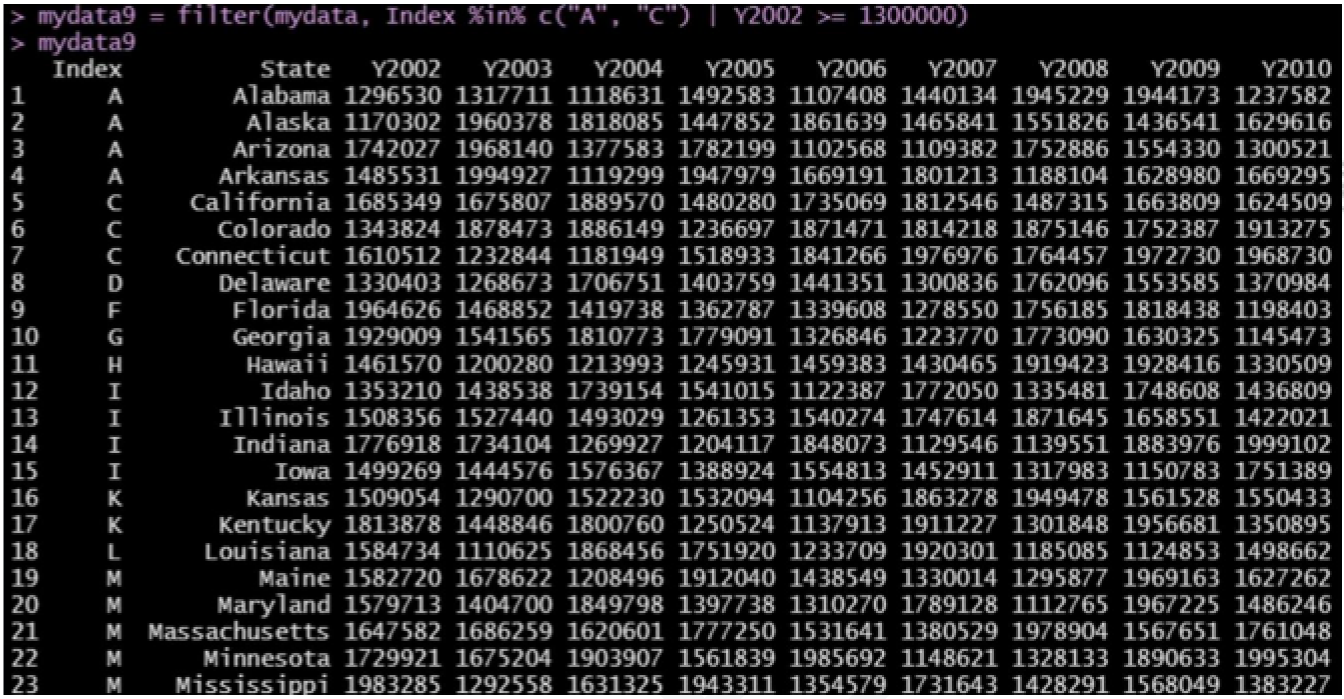
Example 15 : 'OR' Condition in Selection Criteria

7/24

Output

The 'I' denotes OR in the logical condition. It means any of the two conditions.

mydata9 = filter(mydata6, Index %in% c("A", "C") | Y2002 >= 1300000)

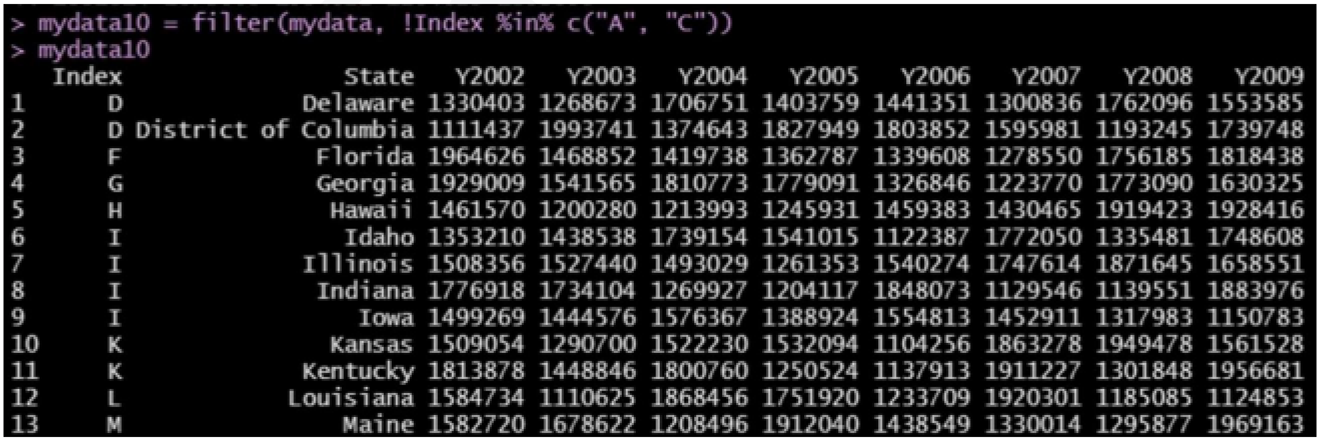


```
> mydata9 = filter(mydata, Index %in% c("A", "C") | Y2002 >= 1300000)
> mydata9
   Index         State   Y2002   Y2003   Y2004   Y2005   Y2006   Y2007   Y2008   Y2009   Y2010
1      A       Alabama 1296530 1317711 1118631 1492583 1107408 1440134 1945229 1944173 1237582
2      A        Alaska 1170302 1960378 1818085 1447852 1861639 1465841 1551826 1436541 1629616
3      A       Arizona 1742027 1968140 1377583 1782199 1102568 1109382 1752886 1554330 1300521
4      A      Arkansas 1485531 1994927 1119299 1947979 1669191 1801213 1188104 1628980 1669295
5      C    California 1685349 1675807 1889570 1480280 1735069 1812546 1487315 1663809 1624509
6      C      Colorado 1343824 1878473 1886149 1236697 1871471 1814218 1875146 1752387 1913275
7      C   Connecticut 1610512 1232844 1181949 1518933 1841266 1976976 1764457 1972730 1968730
8      D      Delaware 1330403 1268673 1706751 1403759 1441351 1300836 1762096 1553585 1370984
9      F       Florida 1964626 1468852 1419738 1362787 1339608 1278550 1756185 1818438 1198403
10     G       Georgia 1929009 1541565 1810773 1779091 1326846 1223770 1773090 1630325 1145473
11     H        Hawaii 1461570 1200280 1213993 1245931 1459383 1430465 1919423 1928416 1330509
12     I         Idaho 1353210 1438538 1739154 1541015 1122387 1772050 1335481 1748608 1436809
13     I      Illinois 1508356 1527440 1493029 1261353 1540274 1747614 1871645 1658551 1422021
14     I       Indiana 1776918 1734104 1269927 1204117 1848073 1129546 1139551 1883976 1999102
15     I          Iowa 1499269 1444576 1576367 1388924 1554813 1452911 1317983 1150783 1751389
16     K        Kansas 1509054 1290700 1522230 1532094 1104256 1863278 1949478 1561528 1550433
17     K      Kentucky 1813878 1448846 1800760 1250524 1137913 1911227 1301848 1956681 1350895
18     L     Louisiana 1584734 1110625 1868456 1751920 1233709 1920301 1185085 1124853 1498662
19     M         Maine 1582720 1678622 1208496 1912040 1438549 1330014 1295877 1969163 1627262
20     M      Maryland 1579713 1404700 1849798 1397738 1310270 1789128 1112765 1967225 1486246
21     M Massachusetts 1647582 1686259 1620601 1777250 1531641 1380529 1978904 1567651 1761048
22     M     Minnesota 1729921 1675204 1903907 1561839 1985692 1148621 1328133 1890633 1995304
23     M   Mississippi 1983285 1292558 1631325 1943311 1354579 1731643 1428291 1568049 1383227
```

Example 16 : NOT Condition

The "!" sign is used to reverse the logical condition.

mydata10 = filter(mydata6, !Index %in% c("A", "C"))

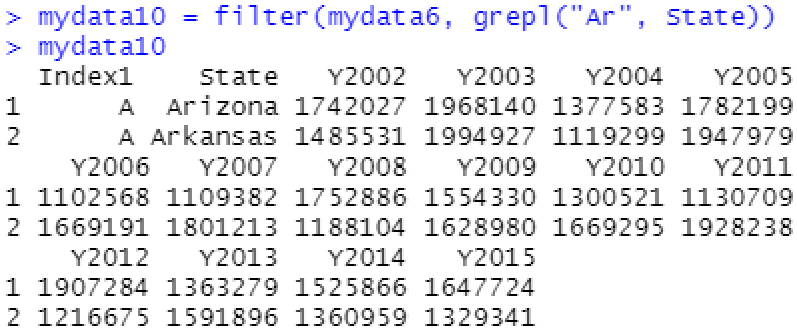


```
> mydata10 = filter(mydata, !Index %in% c("A", "C"))
> mydata10
   Index                State   Y2002   Y2003   Y2004   Y2005   Y2006   Y2007   Y2008   Y2009
1      D             Delaware 1330403 1268673 1706751 1403759 1441351 1300836 1762096 1553585
2      D District of Columbia 1111437 1993741 1374643 1827949 1803852 1595981 1193245 1739748
3      F              Florida 1964626 1468852 1419738 1362787 1339608 1278550 1756185 1818438
4      G              Georgia 1929009 1541565 1810773 1779091 1326846 1223770 1773090 1630325
5      H               Hawaii 1461570 1200280 1213993 1245931 1459383 1430465 1919423 1928416
6      I                Idaho 1353210 1438538 1739154 1541015 1122387 1772050 1335481 1748608
7      I             Illinois 1508356 1527440 1493029 1261353 1540274 1747614 1871645 1658551
8      I              Indiana 1776918 1734104 1269927 1204117 1848073 1129546 1139551 1883976
9      I                 Iowa 1499269 1444576 1576367 1388924 1554813 1452911 1317983 1150783
10     K               Kansas 1509054 1290700 1522230 1532094 1104256 1863278 1949478 1561528
11     K             Kentucky 1813878 1448846 1800760 1250524 1137913 1911227 1301848 1956681
12     L            Louisiana 1584734 1110625 1868456 1751920 1233709 1920301 1185085 1124853
13     M                Maine 1582720 1678622 1208496 1912040 1438549 1330014 1295877 1969163
```

Example 17 : CONTAINS Condition

The **grepl function** is used to search for pattern matching. In the following code, we are looking for records wherein column **state** contains **'Ar'** in their name.

mydata10 = filter(mydata6, grepl("Ar", State))



```
> mydata10 = filter(mydata6, grepl("Ar", State))
> mydata10
  Index1    State   Y2002   Y2003   Y2004   Y2005
1      A  Arizona 1742027 1968140 1377583 1782199
2      A Arkansas 1485531 1994927 1119299 1947979
    Y2006   Y2007   Y2008   Y2009   Y2010   Y2011
1 1102568 1109382 1752886 1554330 1300521 1130709
2 1669191 1801213 1188104 1628980 1669295 1928238
    Y2012   Y2013   Y2014   Y2015
1 1907284 1363279 1525866 1647724
2 1216675 1591896 1360959 1329341
```

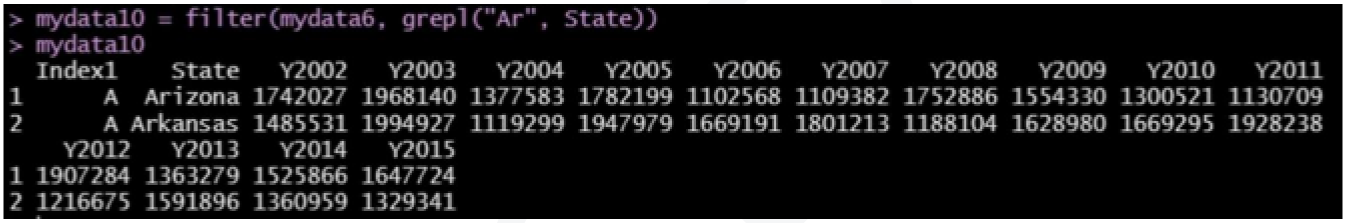
summarise( ) Function

It is used to summarize data.

summarise() syntax : summarise(data , **....**)

data : Data Frame

..... : Summary Functions such as mean, median etc

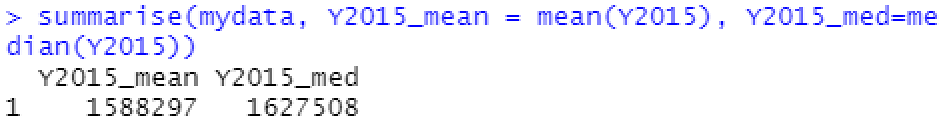


```
> mydata10 = filter(mydata6, grepl("Ar", State))
> mydata10
  Index1    State   Y2002   Y2003   Y2004   Y2005   Y2006   Y2007   Y2008   Y2009   Y2010   Y2011
1      A  Arizona 1742027 1968140 1377583 1782199 1102568 1109382 1752886 1554330 1300521 1130709
2      A Arkansas 1485531 1994927 1119299 1947979 1669191 1801213 1188104 1628980 1669295 1928238
    Y2012   Y2013   Y2014   Y2015
1 1907284 1363279 1525866 1647724
2 1216675 1591896 1360959 1329341
```

Example 18 : Summarize selected variables

In the example below, we are calculating mean and median for the variable Y2015.

summarise(mydata, Y2015_mean = mean(Y2015), Y2015_med=median(Y2015))



```
> summarise(mydata, Y2015_mean = mean(Y2015), Y2015_med=me
dian(Y2015))
  Y2015_mean Y2015_med
1    1588297   1627508
```

Example 19 : Summarize Multiple Variables

In the following example, we are calculating

number of records, mean and median for

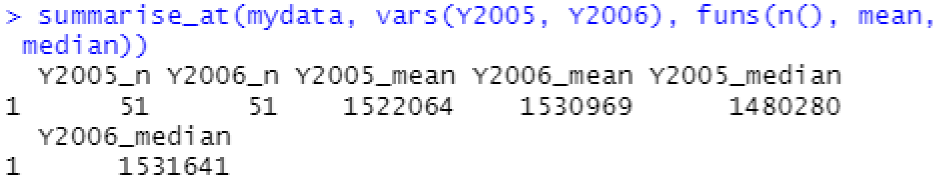
variables Y2005 and Y2006. The **summarise_at** function allows us

to 8/24

select multiple variables by their names.

summarise_at(mydata, vars(Y2005, Y2006), funs(n(), mean, median))

Output



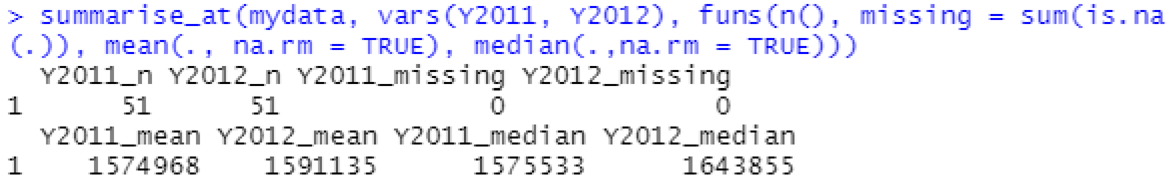
```
> summarise_at(mydata, vars(Y2005, Y2006), funs(n(), mean,
 median))
  Y2005_n Y2006_n Y2005_mean Y2006_mean Y2005_median
1      51      51    1522064    1530969      1480280
  Y2006_median
1      1531641
```

Example 20 : Summarize with Custom Functions

We can also use custom functions in the summarise function. In this case, we are computing the number of records, number of missing values, mean and median for variables Y2011 and Y2012. The **dot (.)** denotes each variables specified in the second argument of the function.

summarise_at(mydata, vars(Y2011, Y2012), funs(n(), missing = **sum(is.na(.)), mean(., na.rm = TRUE), median(.,na.rm = TRUE)))**

Summarize : Output



```
> summarise_at(mydata, vars(Y2011, Y2012), funs(n(), missing = sum(is.na
(.)), mean(., na.rm = TRUE), median(.,na.rm = TRUE)))
  Y2011_n Y2012_n Y2011_missing Y2012_missing
1      51      51             0             0
  Y2011_mean Y2012_mean Y2011_median Y2012_median
1    1574968    1591135      1575533      1643855
```

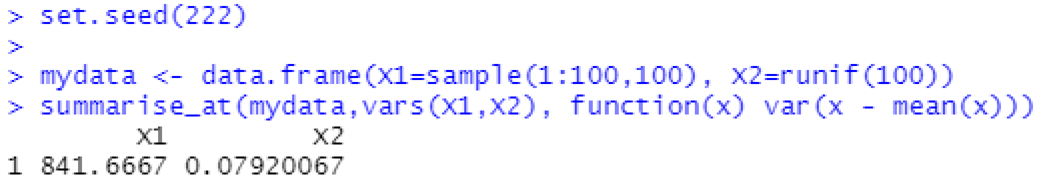
How to apply Non-Standard Functions

Suppose you want to subtract mean from its original value and then calculate variance of it.

set.seed(222)

mydata <- data.frame(X1=sample(1:100,100), X2=runif(100))

summarise_at(mydata,vars(X1,X2), **function(x) var(x - mean(x))**)

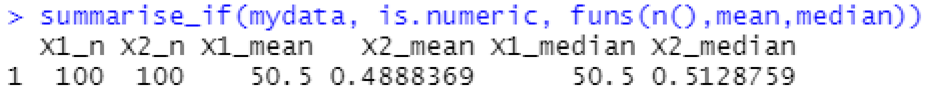


```
> set.seed(222)
> 
> mydata <- data.frame(X1=sample(1:100,100), X2=runif(100))
> summarise_at(mydata,vars(X1,X2), function(x) var(x - mean(x)))
        X1         X2
1 841.6667 0.07920067
```

Example 21 : Summarize all Numeric Variables

The **summarise_if** function allows you to summarize conditionally.

summarise_if(mydata, is.numeric, funs(n(),mean,median))

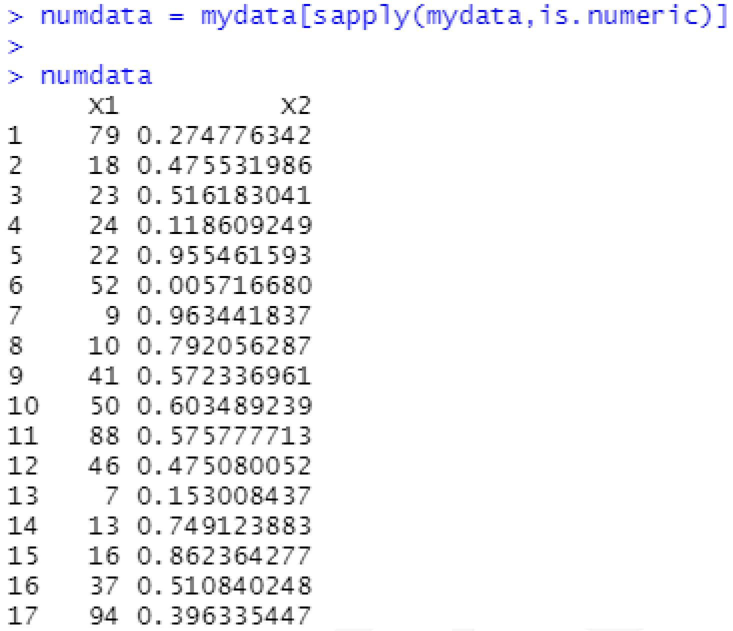


```
> summarise_if(mydata, is.numeric, funs(n(),mean,median))
  X1_n X2_n X1_mean   X2_mean X1_median X2_median
1  100  100    50.5 0.4888369      50.5 0.5128759
```

Alternative Method :

First, store data for all the numeric variables

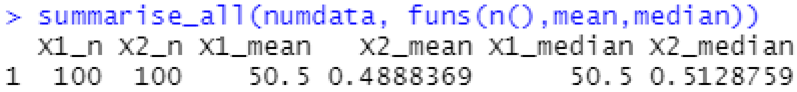
numdata = mydata[sapply(mydata,is.numeric)]



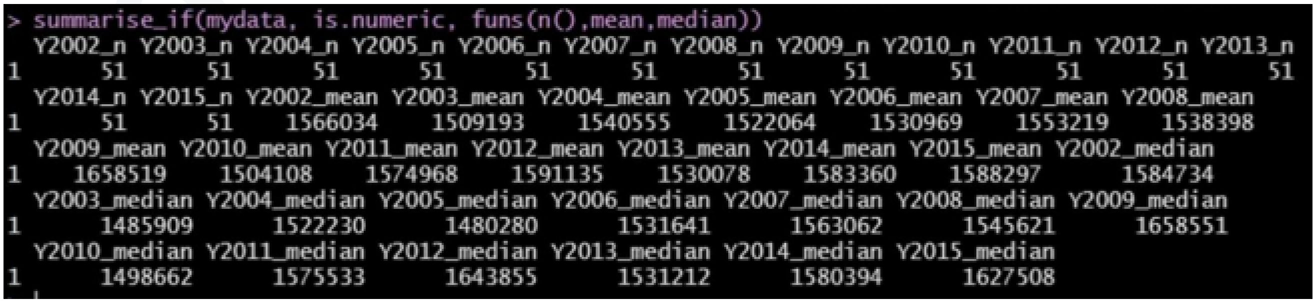
```
> numdata = mydata[sapply(mydata,is.numeric)]
>
> numdata
     X1          X2
1    79 0.274776342
2    18 0.475531986
3    23 0.516183041
4    24 0.118609249
5    22 0.955461593
6    52 0.005716680
7     9 0.963441837
8    10 0.792056287
9    41 0.572336961
10   50 0.603489239
11   88 0.575777713
12   46 0.475080052
13    7 0.153008437
14   13 0.749123883
15   16 0.862364277
16   37 0.510840248
17   94 0.396335447
```

Second, the **summarise_all** function calculates summary statistics for all the columns in a data frame

summarise_all(numdata, funs(n(),mean,median))



```
> summarise_all(numdata, funs(n(),mean,median))
  X1_n X2_n X1_mean   X2_mean X1_median X2_median
1  100  100    50.5 0.4888369      50.5 0.5128759
```



```
> summarise_if(mydata, is.numeric, funs(n(),mean,median))
  Y2002_n Y2003_n Y2004_n Y2005_n Y2006_n Y2007_n Y2008_n Y2009_n Y2010_n Y2011_n Y2012_n Y2013_n
1      51      51      51      51      51      51      51      51      51      51      51      51
  Y2014_n Y2015_n Y2002_mean Y2003_mean Y2004_mean Y2005_mean Y2006_mean Y2007_mean Y2008_mean
1      51      51    1566034    1509193    1540555    1522064    1530969    1553219    1538398
  Y2009_mean Y2010_mean Y2011_mean Y2012_mean Y2013_mean Y2014_mean Y2015_mean Y2002_median
1    1658519    1504108    1574968    1591135    1530078    1583360    1588297      1584734
  Y2003_median Y2004_median Y2005_median Y2006_median Y2007_median Y2008_median Y2009_median
1      1485909      1522230      1480280      1531641      1563062      1545621      1658551
  Y2010_median Y2011_median Y2012_median Y2013_median Y2014_median Y2015_median
1      1498662      1575533      1643855      1531212      1580394      1627508
```

Example 22 : Summarize Factor Variable

We are checking the **number of levels/categories** and **count of missing observations** in a categorical (factor) variable.

```
summarise_all(mydata["Index"], funs(nlevels(.),
nmiss=sum(is.na(.))))
```

nlevels nmiss

1 19 0

arrange() function :

Use : Sort data

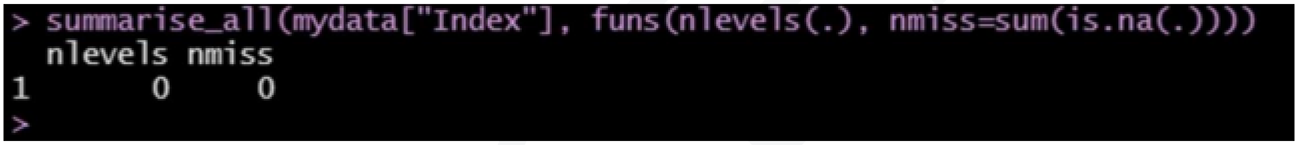
Syntax

```
arrange(data_frame, variable(s)_to_sort)
```

or

data_frame **%>%** arrange(variable(s)_to_sort)

To sort a variable in descending order, use **desc(x)**.



```
> summarise_all(mydata["Index"], funs(nlevels(.), nmiss=sum(is.na(.))))
  nlevels nmiss
1       0     0
>
```

Example 23 : Sort Data by Multiple Variables

The default sorting order of **arrange() function** is ascending. In this example, we are sorting data by multiple variables.

```
arrange(mydata, Index, Y2011)
```

Suppose you need to sort one variable by descending order and other variable by ascending oder.

arrange(mydata, **desc(Index)**, Y2011)

Pipe Operator %>%

It is important to understand the pipe (%>%) operator before knowing the other functions of dplyr package. dplyr utilizes pipe operator from another package **(magrittr)**.

It allows you to write sub-queries like we do it in sql.

Note : All the functions in dplyr package can be used **without** the pipe operator. The question arises **"Why to use pipe operator %>%". The answer is** it lets to wrap multiple functions together with the use of %>%. **Syntax :**

```
filter(data_frame, variable == value)
```

or

```
data_frame %>% filter(variable == value)
```

The %>% is NOT restricted to filter function. It can be used with any function.

Example :

The code below demonstrates the usage of pipe %>% operator. In this example, we are selecting 10 random observations of two variables "Index" "State" from the data frame "mydata".

```
dt = sample_n(select(mydata, Index, State),10)
```

or

```
dt = mydata %>% select(Index, State) %>% sample_n(10)
```

group_by() function :

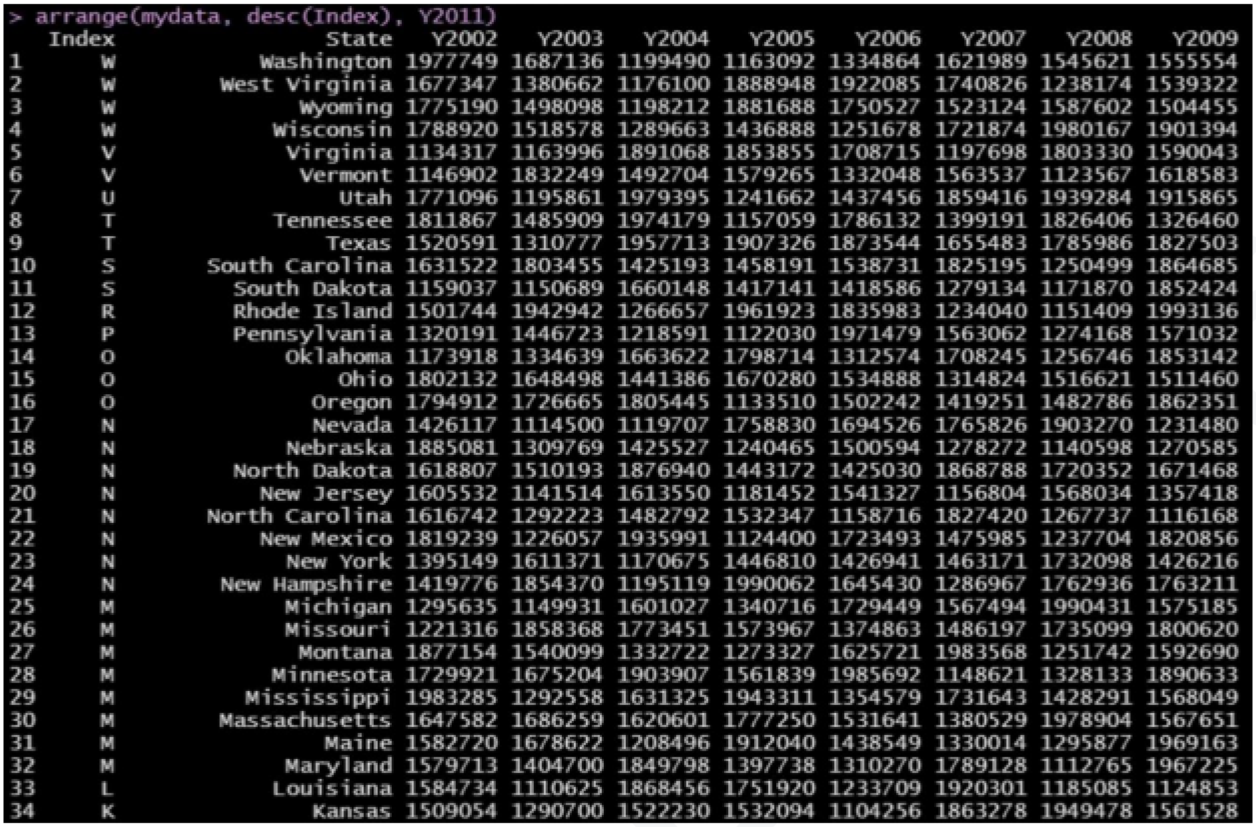
Use : Group data by categorical variable

Syntax :

```
group_by(data, variables)
```

or

```
data %>% group_by(variables)
```



```
> arrange(mydata, desc(Index), Y2011)
   Index                State   Y2002   Y2003   Y2004   Y2005   Y2006   Y2007   Y2008   Y2009
1     W           Washington 1977749 1687136 1199490 1163092 1334864 1621989 1545621 1555554
2     W        West Virginia 1677347 1380662 1176100 1888948 1922085 1740826 1238174 1539322
3     W              Wyoming 1775190 1498098 1198212 1881688 1750527 1523124 1587602 1504455
4     W            Wisconsin 1788920 1518578 1289663 1436888 1251678 1721874 1980167 1901394
5     V             Virginia 1134317 1163996 1891068 1853855 1708715 1197698 1803330 1590043
6     V              Vermont 1146902 1832249 1492704 1579265 1332048 1563537 1123567 1618583
7     U                 Utah 1771096 1195861 1979395 1241662 1437456 1859416 1939284 1915865
8     T            Tennessee 1811867 1485909 1974179 1157059 1786132 1399191 1826406 1326460
9     T                Texas 1520591 1310777 1957713 1907326 1873544 1655483 1785986 1827503
10    S       South Carolina 1631522 1803455 1425193 1458191 1538731 1825195 1250499 1864685
11    S         South Dakota 1159037 1150689 1660148 1417141 1418586 1279134 1171870 1852424
12    R         Rhode Island 1501744 1942942 1266657 1961923 1835983 1234040 1151409 1993136
13    P         Pennsylvania 1320191 1446723 1218591 1122030 1971479 1563062 1274168 1571032
14    O             Oklahoma 1173918 1334639 1663622 1798714 1312574 1708245 1256746 1853142
15    O                 Ohio 1802132 1648498 1441386 1670280 1534888 1314824 1516621 1511460
16    O               Oregon 1794912 1726665 1805445 1133510 1502242 1419251 1482786 1862351
17    N               Nevada 1426117 1114500 1119707 1758830 1694526 1765826 1903270 1231480
18    N             Nebraska 1885081 1309769 1425527 1240465 1500594 1278272 1140598 1270585
19    N         North Dakota 1618807 1510193 1876940 1443172 1425030 1868788 1720352 1671468
20    N           New Jersey 1605532 1141514 1613550 1181452 1541327 1156804 1568034 1357418
21    N       North Carolina 1616742 1292223 1482792 1532347 1158716 1827420 1267737 1116168
22    N           New Mexico 1819239 1226057 1935991 1124400 1723493 1475985 1237704 1820856
23    N             New York 1395149 1611371 1170675 1446810 1426941 1463171 1732098 1426216
24    N        New Hampshire 1419776 1854370 1195119 1990062 1645430 1286967 1762936 1763211
25    M             Michigan 1295635 1149931 1601027 1340716 1729449 1567494 1990431 1575185
26    M             Missouri 1221316 1858368 1773451 1573967 1374863 1486197 1735099 1800620
27    M              Montana 1877154 1540099 1332722 1273327 1625721 1983568 1251742 1592690
28    M            Minnesota 1729921 1675204 1903907 1561839 1985692 1148621 1328133 1890633
29    M          Mississippi 1983285 1292558 1631325 1943311 1354579 1731643 1428291 1568049
30    M        Massachusetts 1647582 1686259 1620601 1777250 1531641 1380529 1978904 1567651
31    M                Maine 1582720 1678622 1208496 1912040 1438549 1330014 1295877 1969163
32    M             Maryland 1579713 1404700 1849798 1397738 1310270 1789128 1112765 1967225
33    L            Louisiana 1584734 1110625 1868456 1751920 1233709 1920301 1185085 1124853
34    K               Kansas 1509054 1290700 1522230 1532094 1104256 1863278 1949478 1561528
```

Example 24 : Summarise Data by

Categorical Variable

We are calculating count and mean of variables Y2011 and Y2012

by variable Index.

```
t = summarise_at(group_by(mydata, Index), vars(Y2011, Y2012),
funs(n(), mean(., na.rm = TRUE)))
```

The above code can also be written like

```
t = mydata %>% group_by(Index) %>%
summarise_at(vars(Y2011:Y2015), funs(n(), mean(., na.rm = TRUE)))
```

do() function :

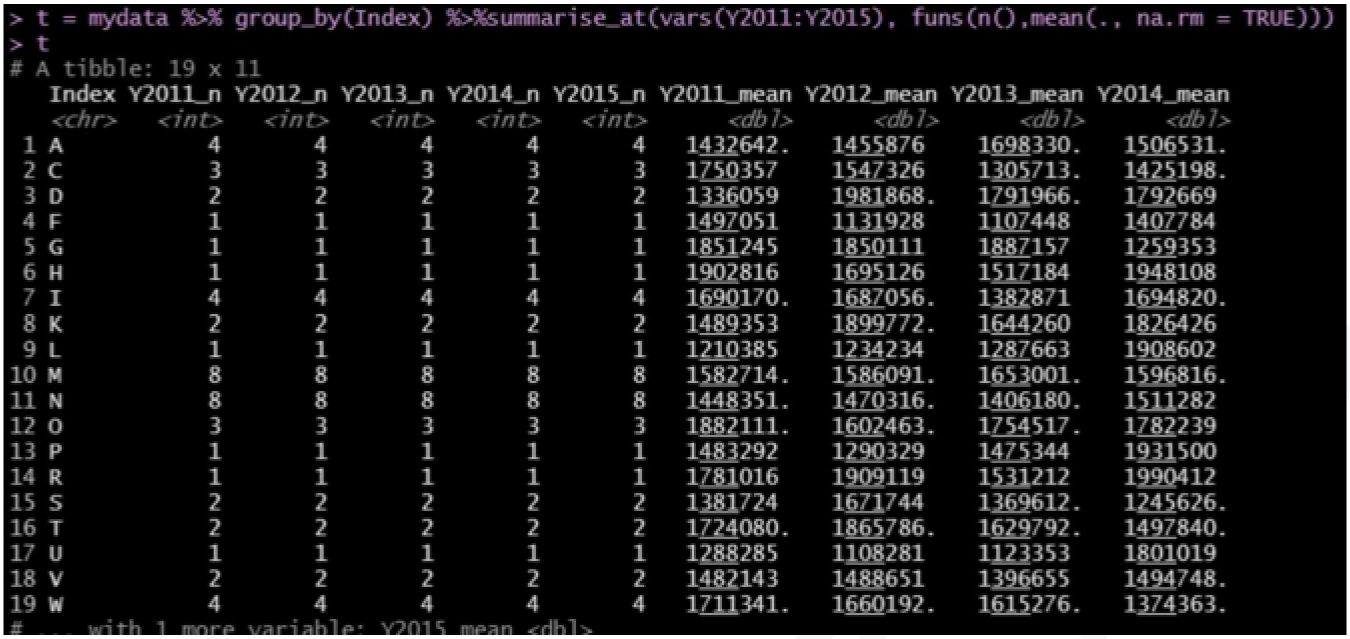
Use : Compute within groups

Syntax :
do(data_frame, expressions_to_apply_to_each_group)

Note : *The* ***dot (.)*** *is required to refer to a data frame.*

12/24

Output



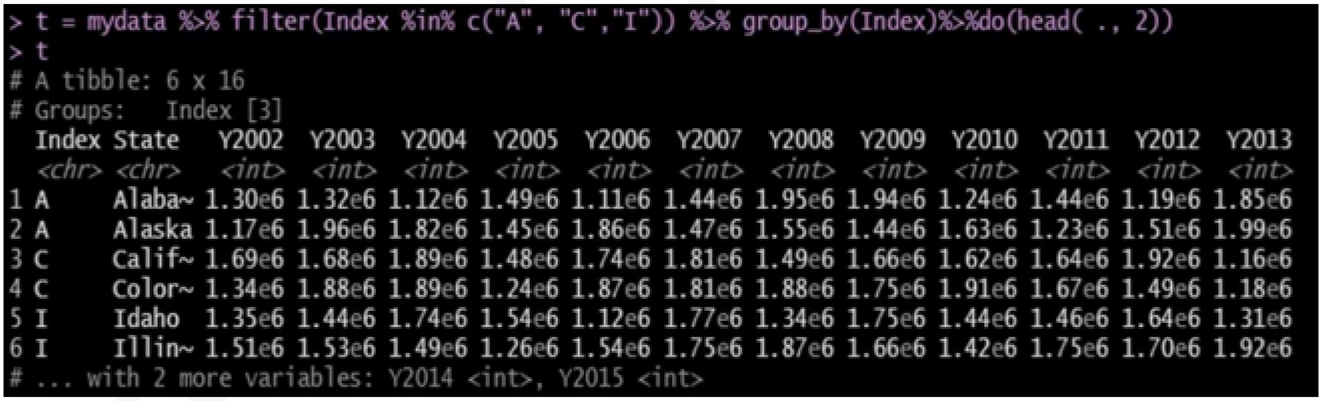
```
> t = mydata %>% group_by(Index) %>%summarise_at(vars(Y2011:Y2015), funs(n(),mean(., na.rm = TRUE)))
> t
# A tibble: 19 x 11
   Index Y2011_n Y2012_n Y2013_n Y2014_n Y2015_n Y2011_mean Y2012_mean Y2013_mean Y2014_mean
   <chr>   <int>   <int>   <int>   <int>   <int>      <dbl>      <dbl>      <dbl>      <dbl>
 1 A           4       4       4       4       4   1432642.    1455876   1698330.   1506531.
 2 C           3       3       3       3       3   1750357     1547326   1305713.   1425198.
 3 D           2       2       2       2       2   1336059     1981868.  1791966.   1792669
 4 F           1       1       1       1       1   1497051     1131928   1107448    1407784
 5 G           1       1       1       1       1   1851245     1850111   1887157    1259353
 6 H           1       1       1       1       1   1902816     1695126   1517184    1948108
 7 I           4       4       4       4       4   1690170.    1687056.  1382871    1694820.
 8 K           2       2       2       2       2   1489353     1899772.  1644260    1826426
 9 L           1       1       1       1       1   1210385     1234234   1287663    1908602
10 M           8       8       8       8       8   1582714.    1586091.  1653001.   1596816.
11 N           8       8       8       8       8   1448351.    1470316.  1406180.   1511282
12 O           3       3       3       3       3   1882111.    1602463.  1754517.   1782239
13 P           1       1       1       1       1   1483292     1290329   1475344    1931500
14 R           1       1       1       1       1   1781016     1909119   1531212    1990412
15 S           2       2       2       2       2   1381724     1671744   1369612.   1245626.
16 T           2       2       2       2       2   1724080.    1865786.  1629792.   1497840.
17 U           1       1       1       1       1   1288285     1108281   1123353    1801019
18 V           2       2       2       2       2   1482143     1488651   1396655    1494748.
19 W           4       4       4       4       4   1711341.    1660192.  1615276.   1374363.
# ... with 1 more variable: Y2015_mean <dbl>
```

Example 25 : Filter Data within a Categorical Variable Suppose you need to pull top 2 rows from 'A', 'C' and 'I' categories of variable Index.

t = mydata %>% filter(Index %in% c("A", "C","I")) %>%

group_by(Index) %>%

do(head(. , 2))



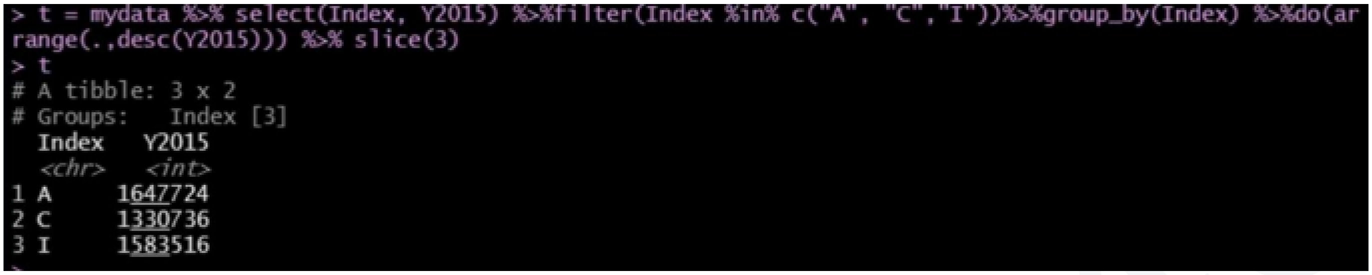
```
> t = mydata %>% filter(Index %in% c("A", "C","I")) %>% group_by(Index)%>%do(head( ., 2))
> t
# A tibble: 6 x 16
# Groups:   Index [3]
  Index State   Y2002  Y2003  Y2004  Y2005  Y2006  Y2007  Y2008  Y2009  Y2010  Y2011  Y2012  Y2013
  <chr> <chr>   <int>  <int>  <int>  <int>  <int>  <int>  <int>  <int>  <int>  <int>  <int>  <int>
1 A     Alaba~ 1.30e6 1.32e6 1.12e6 1.49e6 1.11e6 1.44e6 1.95e6 1.94e6 1.24e6 1.44e6 1.19e6 1.85e6
2 A     Alaska 1.17e6 1.96e6 1.82e6 1.45e6 1.86e6 1.47e6 1.55e6 1.44e6 1.63e6 1.23e6 1.51e6 1.99e6
3 C     Calif~ 1.69e6 1.68e6 1.89e6 1.48e6 1.74e6 1.81e6 1.49e6 1.66e6 1.62e6 1.64e6 1.92e6 1.16e6
4 C     Color~ 1.34e6 1.88e6 1.89e6 1.24e6 1.87e6 1.81e6 1.88e6 1.75e6 1.91e6 1.67e6 1.49e6 1.18e6
5 I     Idaho  1.35e6 1.44e6 1.74e6 1.54e6 1.12e6 1.77e6 1.34e6 1.75e6 1.44e6 1.46e6 1.64e6 1.31e6
6 I     Illin~ 1.51e6 1.53e6 1.49e6 1.26e6 1.54e6 1.75e6 1.87e6 1.66e6 1.42e6 1.75e6 1.70e6 1.92e6
# ... with 2 more variables: Y2014 <int>, Y2015 <int>
```

Example 26 : Selecting 3rd Maximum Value by Categorical Variable We are calculating the third maximum value of variable Y2015 by variable Index. The following code first selects only two variables Index and Y2015. Then it filters the variable Index with 'A', 'C' and 'I' and then it groups the same variable and sorts the variable Y2015 in descending order. At last, it selects the third row.

t = mydata %>% select(Index, Y2015) %>%

filter(Index %in% c("A", "C","I")) %>%

group_by(Index) %>%



```
> t = mydata %>% select(Index, Y2015) %>%filter(Index %in% c("A", "C","I"))%>%group_by(Index) %>%do(ar
range(.,desc(Y2015))) %>% slice(3)
> t
# A tibble: 3 x 2
# Groups:   Index [3]
  Index   Y2015
  <chr>   <int>
1 A     1647724
2 C     1330736
3 I     1583516
```

do(arrange(.,desc(Y2015))) %>% slice(3)

The **slice() function** is used to select rows by position.

Using Window Functions

Like SQL, dplyr uses window functions that are used to subset data within a group. It returns a vector of values. We could use **min_rank() function** that calculates rank in the preceding example,

t = mydata %>% select(Index, Y2015) %>%

filter(Index %in% c("A", "C","I")) %>%

group_by(Index) %>%

filter(**min_rank**(desc(Y2015)) == 3)

Index Y2015

1 A 1647724

2 C 1330736

3 I 1583516

Example 27 : Summarize, Group and Sort Together In this case, we are computing the mean of variables Y2014 and Y2015 by variable Index. Then sort the result by calculated mean variable Y2015. t = mydata %>%

group_by(Index)%>%

summarise(Mean_2014 = mean(Y2014,

na.rm=TRUE), Mean_2015 = mean(Y2015,

na.rm=TRUE)) %>%

arrange(desc(Mean_2015))

mutate() function :

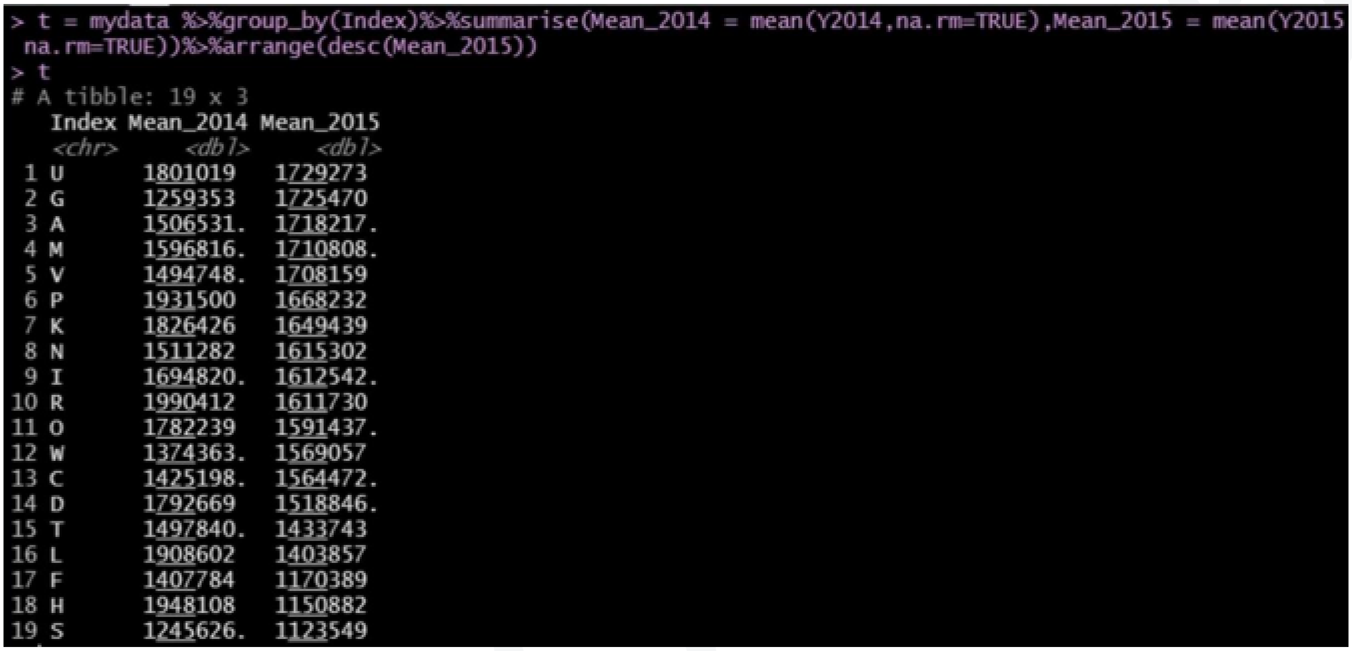
Use : Creates new variables

Syntax :

mutate(data_frame, expression(s))

or

data_frame %>% mutate(expression(s))

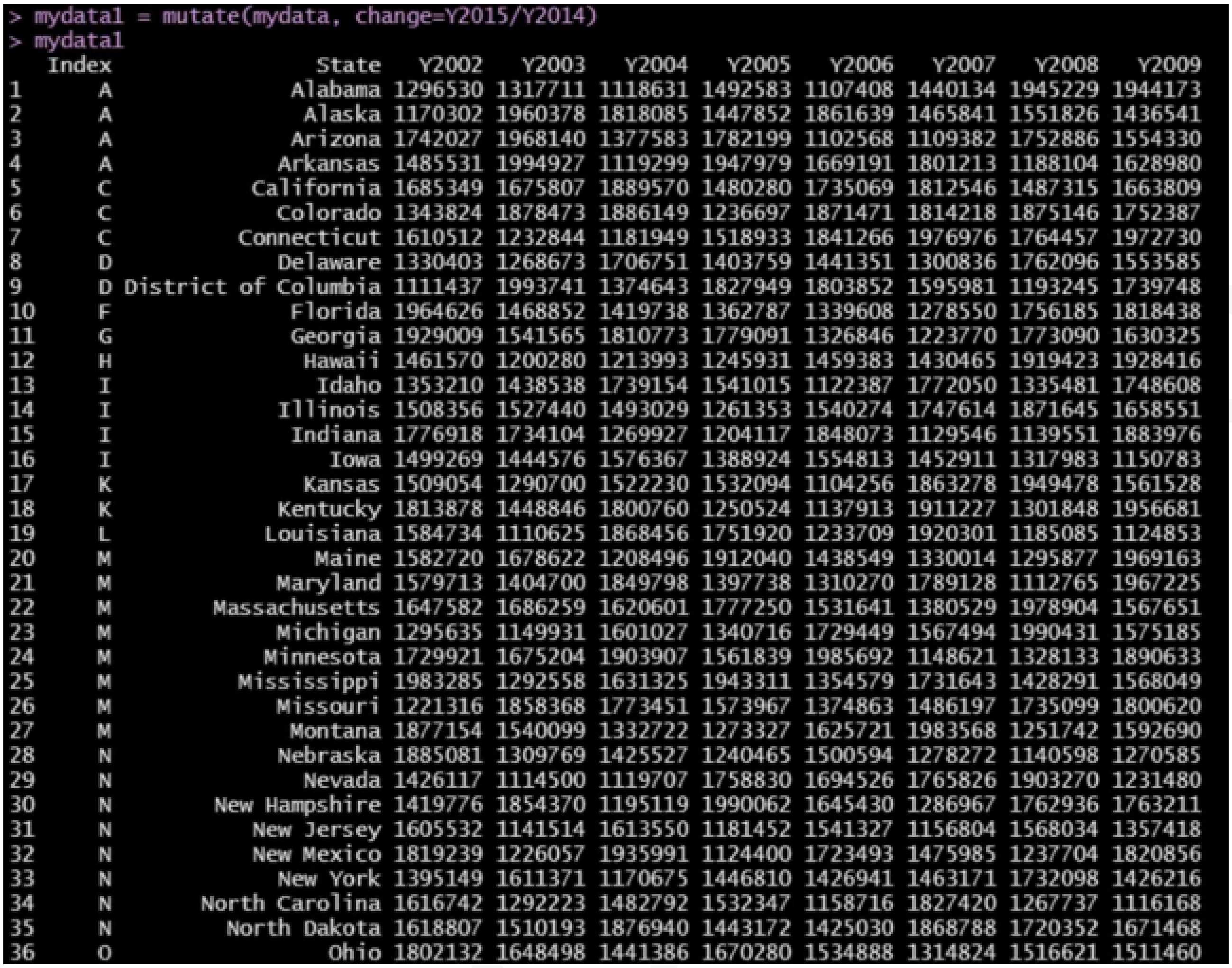


```
> t = mydata %>%group_by(Index)%>%summarise(Mean_2014 = mean(Y2014,na.rm=TRUE),Mean_2015 = mean(Y2015
 na.rm=TRUE))%>%arrange(desc(Mean_2015))
> t
# A tibble: 19 x 3
   Index Mean_2014 Mean_2015
   <chr>     <dbl>     <dbl>
 1 U       1801019   1729273
 2 G       1259353   1725470
 3 A       1506531.  1718217.
 4 M       1596816.  1710808.
 5 V       1494748.  1708159
 6 P       1931500   1668232
 7 K       1826426   1649439
 8 N       1511282   1615302
 9 I       1694820.  1612542.
10 R       1990412   1611730
11 O       1782239   1591437.
12 W       1374363.  1569057
13 C       1425198.  1564472.
14 D       1792669   1518846.
15 T       1497840.  1433743
16 L       1908602   1403857
17 F       1407784   1170389
18 H       1948108   1150882
19 S       1245626.  1123549
```

Example 28 : Create a new variable

The following code calculates the division of Y2015 by Y2014 and names it "change".

mydata1 = mutate(mydata, change=Y2015/Y2014)



```
> mydata1 = mutate(mydata, change=Y2015/Y2014)
> mydata1
   Index                State   Y2002   Y2003   Y2004   Y2005   Y2006   Y2007   Y2008   Y2009
1      A              Alabama 1296530 1317711 1118631 1492583 1107408 1440134 1945229 1944173
2      A               Alaska 1170302 1960378 1818085 1447852 1861639 1465841 1551826 1436541
3      A              Arizona 1742027 1968140 1377583 1782199 1102568 1109382 1752886 1554330
4      A             Arkansas 1485531 1994927 1119299 1947979 1669191 1801213 1188104 1628980
5      C           California 1685349 1675807 1889570 1480280 1735069 1812546 1487315 1663809
6      C             Colorado 1343824 1878473 1886149 1236697 1871471 1814218 1875146 1752387
7      C          Connecticut 1610512 1232844 1181949 1518933 1841266 1976976 1764457 1972730
8      D             Delaware 1330403 1268673 1706751 1403759 1441351 1300836 1762096 1553585
9      D District of Columbia 1111437 1993741 1374643 1827949 1803852 1595981 1193245 1739748
10     F              Florida 1964626 1468852 1419738 1362787 1339608 1278550 1756185 1818438
11     G              Georgia 1929009 1541565 1810773 1779091 1326846 1223770 1773090 1630325
12     H               Hawaii 1461570 1200280 1213993 1245931 1459383 1430465 1919423 1928416
13     I                Idaho 1353210 1438538 1739154 1541015 1122387 1772050 1335481 1748608
14     I             Illinois 1508356 1527440 1493029 1261353 1540274 1747614 1871645 1658551
15     I              Indiana 1776918 1734104 1269927 1204117 1848073 1129546 1139551 1883976
16     I                 Iowa 1499269 1444576 1576367 1388924 1554813 1452911 1317983 1150783
17     K               Kansas 1509054 1290700 1522230 1532094 1104256 1863278 1949478 1561528
18     K             Kentucky 1813878 1448846 1800760 1250524 1137913 1911227 1301848 1956681
19     L            Louisiana 1584734 1110625 1868456 1751920 1233709 1920301 1185085 1124853
20     M                Maine 1582720 1678622 1208496 1912040 1438549 1330014 1295877 1969163
21     M             Maryland 1579713 1404700 1849798 1397738 1310270 1789128 1112765 1967225
22     M        Massachusetts 1647582 1686259 1620601 1777250 1531641 1380529 1978904 1567651
23     M             Michigan 1295635 1149931 1601027 1340716 1729449 1567494 1990431 1575185
24     M            Minnesota 1729921 1675204 1903907 1561839 1985692 1148621 1328133 1890633
25     M          Mississippi 1983285 1292558 1631325 1943311 1354579 1731643 1428291 1568049
26     M             Missouri 1221316 1858368 1773451 1573967 1374863 1486197 1735099 1800620
27     M              Montana 1877154 1540099 1332722 1273327 1625721 1983568 1251742 1592690
28     N             Nebraska 1885081 1309769 1425527 1240465 1500594 1278272 1140598 1270585
29     N               Nevada 1426117 1114500 1119707 1758830 1694526 1765826 1903270 1231480
30     N        New Hampshire 1419776 1854370 1195119 1990062 1645430 1286967 1762936 1763211
31     N           New Jersey 1605532 1141514 1613550 1181452 1541327 1156804 1568034 1357418
32     N           New Mexico 1819239 1226057 1935991 1124400 1723493 1475985 1237704 1820856
33     N             New York 1395149 1611371 1170675 1446810 1426941 1463171 1732098 1426216
34     N       North Carolina 1616742 1292223 1482792 1532347 1158716 1827420 1267737 1116168
35     N         North Dakota 1618807 1510193 1876940 1443172 1425030 1868788 1720352 1671468
36     O                 Ohio 1802132 1648498 1441386 1670280 1534888 1314824 1516621 1511460
```

Example 29 : Multiply all the variables by 1000

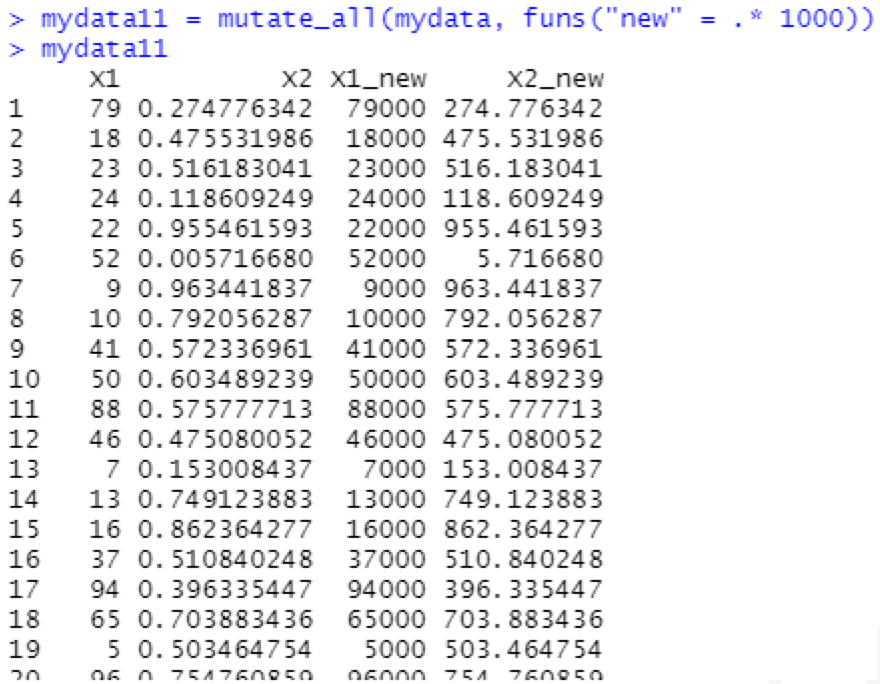
It creates new variables and name them with suffix

"_new". 14/24

Output

Output

mydata11 = mutate_all(mydata, funs("new" = .* 1000))



```
> mydata11 = mutate_all(mydata, funs("new" = .* 1000))
> mydata11
     X1          X2 X1_new     X2_new
1    79 0.274776342  79000 274.776342
2    18 0.475531986  18000 475.531986
3    23 0.516183041  23000 516.183041
4    24 0.118609249  24000 118.609249
5    22 0.955461593  22000 955.461593
6    52 0.005716680  52000   5.716680
7     9 0.963441837   9000 963.441837
8    10 0.792056287  10000 792.056287
9    41 0.572336961  41000 572.336961
10   50 0.603489239  50000 603.489239
11   88 0.575777713  88000 575.777713
12   46 0.475080052  46000 475.080052
13    7 0.153008437   7000 153.008437
14   13 0.749123883  13000 749.123883
15   16 0.862364277  16000 862.364277
16   37 0.510840248  37000 510.840248
17   94 0.396335447  94000 396.335447
18   65 0.703883436  65000 703.883436
19    5 0.503464754   5000 503.464754
20   96 0.754760859  96000 754.760859
```

The output shown in the

The image above is truncated due to a high number of variables.

Note - The above code returns the following error messages -

Warning messages:

1: In Ops.factor(c(1L, 1L, 1L, 1L, 2L, 2L, 2L, 3L, 3L, 4L, 5L, 6L, :

µ¶ QRW PHDQLQJIXO IRU IDFWRUV

,Q 2SV IDFWRU

µ¶ QRW PHDQLQJIXO IRU IDFWRUV

It implies you are multiplying 1000 to string(character) values which are stored as factor variables. These variables are 'Index', 'State'. It does not make sense to apply multiplication operations on character variables. For these two variables, it creates newly created variables which contain only NA.

Solution : See **Example 45** -Apply multiplication on only numeric variables

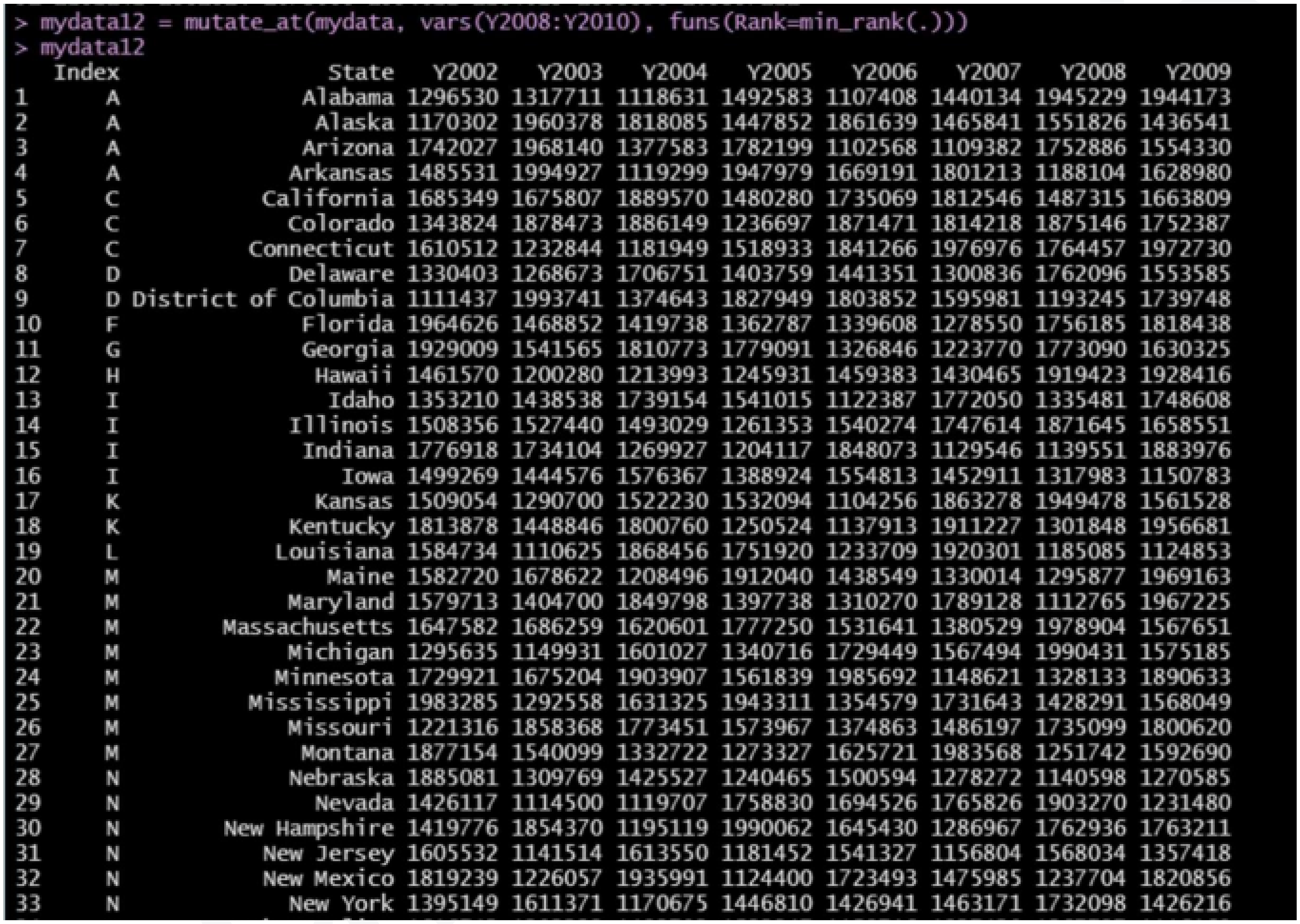
Example 30 : Calculate Rank for Variables

Suppose you need to calculate rank for variables Y2008 to Y2010.

mydata12 = mutate_at(mydata, vars(Y2008:Y2010),funs(Rank=min_rank(.)))

By default, **min_rank()** assigns 1 to the smallest value and high number to the largest value. In case, you need to assign rank 1 to the largest value of a variable, use **min_rank(desc(.))**

mydata13 = mutate_at(mydata, vars(Y2008:Y2010), funs(Rank=min_rank(desc(.))))



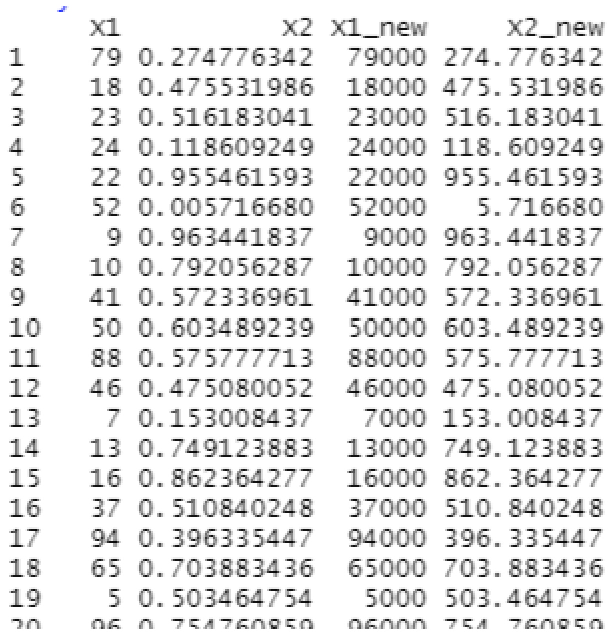
```
> mydata12 = mutate_at(mydata, vars(Y2008:Y2010), funs(Rank=min_rank(.)))
> mydata12
   Index                State   Y2002   Y2003   Y2004   Y2005   Y2006   Y2007   Y2008   Y2009
1      A              Alabama 1296530 1317711 1118631 1492583 1107408 1440134 1945229 1944173
2      A               Alaska 1170302 1960378 1818085 1447852 1861639 1465841 1551826 1436541
3      A              Arizona 1742027 1968140 1377583 1782199 1102568 1109382 1752886 1554330
4      A             Arkansas 1485531 1994927 1119299 1947979 1669191 1801213 1188104 1628980
5      C           California 1685349 1675807 1889570 1480280 1735069 1812546 1487315 1663809
6      C             Colorado 1343824 1878473 1886149 1236697 1871471 1814218 1875146 1752387
7      C          Connecticut 1610512 1232844 1181949 1518933 1841266 1976976 1764457 1972730
8      D             Delaware 1330403 1268673 1706751 1403759 1441351 1300836 1762096 1553585
9      D District of Columbia 1111437 1993741 1374643 1827949 1803852 1595981 1193245 1739748
10     F              Florida 1964626 1468852 1419738 1362787 1339608 1278550 1756185 1818438
11     G              Georgia 1929009 1541565 1810773 1779091 1326846 1223770 1773090 1630325
12     H               Hawaii 1461570 1200280 1213993 1245931 1459383 1430465 1919423 1928416
13     I                Idaho 1353210 1438538 1739154 1541015 1122387 1772050 1335481 1748608
14     I             Illinois 1508356 1527440 1493029 1261353 1540274 1747614 1871645 1658551
15     I              Indiana 1776918 1734104 1269927 1204117 1848073 1129546 1139551 1883976
16     I                 Iowa 1499269 1444576 1576367 1388924 1554813 1452911 1317983 1150783
17     K               Kansas 1509054 1290700 1522230 1532094 1104256 1863278 1949478 1561528
18     K             Kentucky 1813878 1448846 1800760 1250524 1137913 1911227 1301848 1956681
19     L            Louisiana 1584734 1110625 1868456 1751920 1233709 1920301 1185085 1124853
20     M                Maine 1582720 1678622 1208496 1912040 1438549 1330014 1295877 1969163
21     M             Maryland 1579713 1404700 1849798 1397738 1310270 1789128 1112765 1967225
22     M        Massachusetts 1647582 1686259 1620601 1777250 1531641 1380529 1978904 1567651
23     M             Michigan 1295635 1149931 1601027 1340716 1729449 1567494 1990431 1575185
24     M            Minnesota 1729921 1675204 1903907 1561839 1985692 1148621 1328133 1890633
25     M          Mississippi 1983285 1292558 1631325 1943311 1354579 1731643 1428291 1568049
26     M             Missouri 1221316 1858368 1773451 1573967 1374863 1486197 1735099 1800620
27     M              Montana 1877154 1540099 1332722 1273327 1625721 1983568 1251742 1592690
28     N             Nebraska 1885081 1309769 1425527 1240465 1500594 1278272 1140598 1270585
29     N               Nevada 1426117 1114500 1119707 1758830 1694526 1765826 1903270 1231480
30     N        New Hampshire 1419776 1854370 1195119 1990062 1645430 1286967 1762936 1763211
31     N           New Jersey 1605532 1141514 1613550 1181452 1541327 1156804 1568034 1357418
32     N           New Mexico 1819239 1226057 1935991 1124400 1723493 1475985 1237704 1820856
33     N             New York 1395149 1611371 1170675 1446810 1426941 1463171 1732098 1426216
```

Example 31 : Select State that generated highest income among the variable 'Index'

out = mydata %>% group_by(Index) %>%

filter(min_rank(desc(Y2015)) == 1) %>%

select(Index, State, Y2015)



```
     x1          x2 x1_new     x2_new
1    79 0.274776342  79000 274.776342
2    18 0.475531986  18000 475.531986
3    23 0.516183041  23000 516.183041
4    24 0.118609249  24000 118.609249
5    22 0.955461593  22000 955.461593
6    52 0.005716680  52000   5.716680
7     9 0.963441837   9000 963.441837
8    10 0.792056287  10000 792.056287
9    41 0.572336961  41000 572.336961
10   50 0.603489239  50000 603.489239
11   88 0.575777713  88000 575.777713
12   46 0.475080052  46000 475.080052
13    7 0.153008437   7000 153.008437
14   13 0.749123883  13000 749.123883
15   16 0.862364277  16000 862.364277
16   37 0.510840248  37000 510.840248
17   94 0.396335447  94000 396.335447
18   65 0.703883436  65000 703.883436
19    5 0.503464754   5000 503.464754
20   96 0.754760859  96000 754.760859
```